LOAN DOCUMENT

PHOTOGRAPH THIS SHEET

DTIC ACCESSION NUMBER

LEVEL

INVENTORY 6

AFRL-IF-WP-TM-2004-1535

DOCUMENT IDENTIFICATION

Jun 2003

DISTRIBUTION STATEMENT A
Approved for Public Release
Distribution Unlimited

DISTRIBUTION STATEMENT

ACCESSION FOR	
NTIS GRA&I	[✓]
DTIC TRAC	☐
UNANNOUNCED	☐
JUSTIFICATION	
BY	
DISTRIBUTION/	
AVAILABILITY CODES	

DISTRIBUTION	AVAILABILITY AND/OR SPECIAL	
A-1		

DISTRIBUTION STAMP

DATE ACCESSIONED

DATE RETURNED

20050223 136

DATE RECEIVED IN DTIC

REGISTERED OR CERTIFIED NUMBER

PHOTOGRAPH THIS SHEET AND RETURN TO DTIC-FDAC

HANDLE WITH CARE

DTIC FORM JUN 90 70A

DOCUMENT PROCESSING SHEET

PREVIOUS EDITIONS MAY BE USED UNTIL STOCK IS EXHAUSTED.

LOAN DOCUMENT

AFRL-IF-WP-TM-2004-1535

DEVELOPMENT, EXPLOITATION, AND TRANSITION OF COMPUTER AIDED ENGINEERING (CAE) TOOLS

Dr. Harold W. Carter, Ed.

University of Cincinnati
Department of Electrical and Computer Engineering and Computer Science
ML 30
P.O. Box 210030
Cincinnati, OH 45221-0030

JUNE 2003

Final Report for 02 November 1997 – 06 October 2002

STINFO FINAL REPORT

INFORMATION DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7334

NOTICE

[illegible]

Dr. Robert L. Ewing
Project Engineer
Embedded Information Systems Branch
Information Directorate

[illegible]

James S. Williamson
Chief
Embedded Information Systems Branch
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE ***(DD-MM-YY)*** June 2003	**2. REPORT TYPE** Final	**3. DATES COVERED** ***(From - To)*** 11/02/1997 – 10/06/2002

4. TITLE AND SUBTITLE DEVELOPMENT, EXPLOITATION, AND TRANSITION OF COMPUTER AIDED ENGINEERING (CAE) TOOLS	**5a. CONTRACT NUMBER** F33615-96-2-1945
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER 62204F
6. AUTHOR(S) Dr. Harold W. Carter, Ed.	**5d. PROJECT NUMBER** 6096
	5e. TASK NUMBER 40
	5f. WORK UNIT NUMBER 28
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Cincinnati Department of Electrical and Computer Engineering and Computer Science ML 30 P.O. Box 210030 Cincinnati, OH 45221-0030	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Information Directorate Air Force Research Laboratory Air Force Materiel Command Wright-Patterson AFB, OH 45433-7334	**10. SPONSORING/MONITORING AGENCY ACRONYM(S)** AFRL/IFTA
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S) AFRL-IF-WP-TM-2004-1535

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
Documents contained in this technical memo are in the public domain.

14. ABSTRACT
This report documents research results from 22 tasks related to microelectronic modeling and software design aids conducted by researchers at six universities in Ohio and Michigan. Tasks include CMOS-based microwave component design and fabrication, parallel and mixed-signal VHDL and VBHDL-AMS simulator algorithms, conversion tools for VHDL-AMS models, System-on-a-Chip methods, VHDL-AMS modeling of operational amplifiers, mechatronic component design analysis, video/audio data compression using wavelets, and design methods for mixed analog/digital circuits.

15. SUBJECT TERMS
3D MMIC, VHDL, VHDL-AMS, mixed-signal design, parallel/distributed simulation, system-on-a-chip, adaptive filtering, mechatronic design, opamp design, circuit modeling

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT:**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** (Monitor)
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	SAR	358	Dr. Robert L. Ewing **19b. TELEPHONE NUMBER** *(Include Area Code)* (937) 255-6653 x3592

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18

Table of Contents

TASK 1

Java Development Scripts for VHDL/VHDL-AMS Formal Verification Parser

Principal Investigator: Prof. Krishnaprasad Thirunarayan
Institution: Wright State University, Dayton, Ohio
Period of Performance: May 21, 1997 to April 20, 1998

VHDL-AMS Hardware Design Browser

Tianrong Hu and Krishnaprasad Thirunarayan
Department of Computer Science and Engineering
Wright State University, Dayton, OH-45435.

1. Introduction

The VHDL-AMS design browser is a tool for inspecting digital designs written in VHDL-AMS, a superset of VHDL-93. Because VHDL designs can be large and complex, plain text searches are unwieldy and insufficient for determining the structure and organization of the designs. Originally, a VHDL-93 Hardware Description Browser for intelligently retrieving information from VHDL designs was first developed by L.DeBrock and implemented using TCL/TK. In order for the design browser to be usable in both the UNIX and the Windows95/NT environment, the browser has been rewritten in Java.

1.1 Overall Organization

The VHDL Browser consists of a GUI written in Java and a search engine written in SWI-Prolog. The GUI process communicates with the search engine in a master-slave configuration with the GUI acting as the master and the search engine as the slave. The GUI interacts with the Prolog search engine to shield the user from Prolog specific details. The GUI directs the querying process by presenting the user with a set of menus of frequently used queries.

The VHDL Browser provides the following basic functionality:

- Load, parse, and pretty-print VHDL files.
- List all entities, architectures, packages, configurations, user-defined data types, routines, files, components, variables, constants, signals, etc.
- View source code with documentation of a given entity, architecture, package, configuration, user-defined data type and routine.
- Locate definitions for user-defined data types and routines, routine calls, files, components, variables, signals, constants, etc.
- Show the component hierarchy of an entity and determine if X is a subcomponent of Y.

2. Design and Implementation of Java-based GUI

This section describes the overall design and Java specific details for implementing the browser.

2.1 Design of the GUI

As can be seen in Figure 1, the GUI is setup in four steps:

(i) Launch Prolog process (search engine) and obtain its input stream and output stream
(ii) Set up GUI to wait for user actions
(iii) Each query action involves GUI sending the query to and fetching the response from the Prolog process
(iv) GUI shows the results to the user

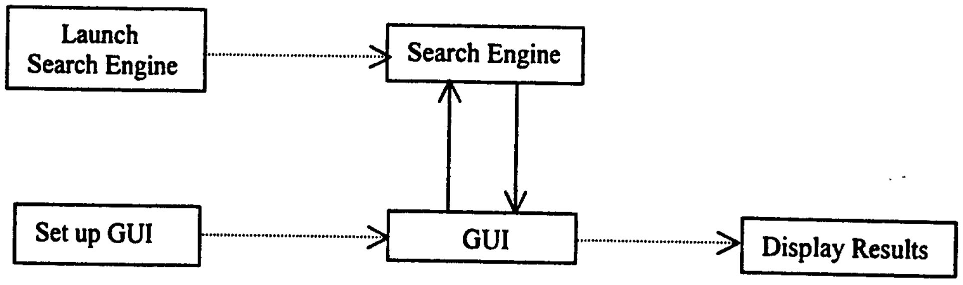


Figure 1: GUI Architecture

Part (i) involves issuing the correct command to launch the Prolog process and obtaining its input stream and output stream to set up the communication with GUI.

Part (ii) involves instantiating the GUI and preparing the GUI for user actions, namely setting up ActionListeners for all the functional components inside the GUI. The GUI components can be divided into 6 groups according to their actions, that is, File, List, View Source, View Doc, Locate, and Associations.

Part (iii) accomplishes the IPC (inter-process communication) between the GUI and the search engine. Each user action triggers the GUI based on Part (ii). After the GUI obtains enough information from the user, the GUI composes the corresponding Prolog query and sends it to the search engine. Once the search engine finishes the query processing, it writes the query result back to the GUI. To match the query with the corresponding response, the GUI maintains a queue of queries in a chronological order. Thus the GUI fetches a query from the queue for each response. Because the search engine processes queries one by one and sends out the responses in the same order as the queries are sent in, there should be no confusion about the correspondence between the queries in the queue and its responses. On the other hand, the query results are continuously sent back, it is impossible for GUI to figure out where the start and the end of a specific query results are. So the result is embedded between a start marker and an end marker.

Part (iv) displays the query results. According to the 6 groups of component actions, there are 6 groups of result windows, which should be placed in 6 different positions to make the user easily organize and analyze the results.

2.2 Implementation of the GUI

This section describes the details of communication between Java and Prolog processes and the menus setup for invoking the various canned queries.

2.2.1 Launch the Search Engine

In Java, a subprocess is created using method exec() of class Runtime in main process. The argument of exec() is a specified system command. Here this command is used to launch the Prolog process. And the command in UNIX environment (pl –x ...) is different from the one in Windows environment (plcon –x ...).

```
Runtime r = Runtime.getRuntime();
Process p_prolog = null;
String cmd_prolog = "vhdl97_parser -g true -f search_engine.pro";
if( System.getProperty("os.name").equals("Solaris"))
        cmd_prolog = "pl -x " + cmd_prolog;
else
        cmd_prolog = "plcon -x " + cmd_prolog;
p_prolog = r.exec(cmd_prolog);
```

The main process controls the subprocess launched by exec(), getInputStream() connects the input stream of the main process to the normal output of the subprocess and getOutputStream() connects the output stream of the main process to the normal input of the subprocess. Thus, the communication between the main process (GUI) and the subprocess (search engine) is set up. The GUI can send information to the search engine using dout_prolog and get information from the search engine using din_prolog.

```
din_prolog = new BufferedReader
                (new InputStreamReader(p_prolog.getInputStream()));
dout_prolog = new DataOutputStream(p_prolog.getOutputStream());
```

2.2.2 Set up the GUI

First, create an instance of GUI to interact with the user. The GUI constructor is shown below:

```
Gui()
{
        super("VHDL Design Browser");
        cmd_list = new Vector();
        file_list = new Vector();

        Font font_menu = new Font("SansSerif", Font.BOLD, 14);
        font = new Font("SansSerif", Font.BOLD, 10);
        setFont(font);

        MenuBar menubar = new MenuBar();
        menubar.setFont(font_menu);

        menubar.add( m_file = new Menu_File() );
        menubar.add( m_list = new Menu_List() );
        menubar.add( m_view = new Menu_View() );
        menubar.add( m_locate = new Menu_Locate() );
        menubar.add( m_assoc = new Menu_Assoc() );
        menubar.add( m_help = new Menu_Help() );
        setMenuBar(menubar);

        add("North", fp = new File_Panel());
        add("Center", bp  = new Browse_Panel());

        pack();
        show();
        setSize(460, 360);
        setLocation(0, 0);
    }
```

The GUI constructor sets up the functional components: Menu_File, Menu_List, Menu_View, Menu_Locate, Menu_Assoc, Menu_Help and File_Panel to wait for the user actions and component Browse_Panel to record user actions. These components are defined as follows:

```
class Menu_File extends Menu { }
class Menu_List extends Menu { }
class Menu_View extends Menu { }
class Menu_Locate extends Menu { }
class Menu_Assoc extends Menu { }
class Menu_Help extends Menu { }

class File_Panel extends Panel { }
class Browse_Panel extends Panel { }
```

The GUI constructor also initializes two Vecotors: cmd_list (the queue for storing query commands) and file_list (a list for storing loaded files).

Once the GUI is instantiated, each functional component needs to be explicitly assigned an ActionListener to respond to the user action. Example cases follow:

```
// Action of "Load File(.vhdl)" in "File" menu
m_file.item_load.addActionListener (new GuiActionListener(gui, "load_file"));

//Action of "Entities" in "List" menu
m_list.item_entities.addActionListener (new GuiActionListener(gui, "list_entities"));

//Action of "Show Source Code of" "Entity" in "View" menu
```

```
m_view.item_entity.addActionListener (new GuiActionListener
                                        (gui, "view_source_entity"));

//Action of "Show Source Code and Doc of" "Entity" in "View" menu
m_view.item_doc_entity.addActionListener (new GuiActionListener
                                        (gui, "view_source_doc_entity"));

//Action of "UserDefinedDataTypeDefn" in "Locate" menu
m_locate.item_datatype.addActionListener (new GuiActionListener
                                        (gui, "locate_datatype"));

//Action of "ComponentHierarchy" in "Associations" menu
m_assoc.item_hierarchy.addActionListener (new GuiActionListener
                                        (gui, "assoc_hierarchy"));

//Action of "FileQueries" in "Help" menu
m_help.item_file.addActionListener (new GuiActionListener(gui, "help_file"));
```

As it can be seen, all the ActionListeners are instances of an universal ActionListener (GuiActionListener), which needs two arguments, one is an instance of GUI, the other is the name of the method defined in GUI for handling the specific action.

2.2.3 Communication between GUI and Search Engine

Sending the Prolog query to the search engine:

```
public void load_file()
{       ...
        String file_path = fd.getDirectory() + fd.getFile();
        ...
        // make sure file separator is always '/' in Prolog file path
        String mes = "java_vhdl_read('" + file_path.replace('\\','/') + "').";

        cmd_list.addElement("Load File");
        ...
        write(mes);
        ...
}

public void list_entities()
{
        addcmd("List All Entities");
        write("list_all_Entities(_).");
}
```

Fetching the response from the search engine:

```
while(true)
{
        //only packed message("start"..."end") shows up
        while(!(mes_prolog = din_prolog.readLine()).equals("start")) ;

        command = (String) cmd_list.elementAt(0);
        cmd_list.removeElementAt(0);

        if(command.equals("Load File"))                 {...}
        ...
        else if(command.equals("List All Entities"))    {...}
        ...
}
```

As discussed, load_file() method is invoked based on the user action. In load_file(), first, the Prolog query is composed, then command "Load File" is added to the queue cmd_list, and the query is sent to the search engine. In the meantime, the GUI keeps waiting for messages from the search engine and only the messages beginning with "start" are accepted by GUI for later processing. The other messages are ignored. Obviously the accepted message is the query result corresponding to the command fetched from cmd_list, which is "Load File" in this case.

Likewise, list_entities() matches command.equals("List All Entities").

2.2.4 Results Display in GUI

The functions for displaying the query results are listed below:

```
static void list_all_results(String cmd, Color clr, String proc, Gui g)
                                                        throws Exception { }
static void view_results(String command, Color clr) throws Exception { }
static void view_results2(String command, Color clr) throws Exception { }
static void view_doc_results(String command, Color clr) throws Exception { }
static void view_doc_results2(String command, Color clr) throws Exception { }
static void locate_results(String command, Color clr) throws Exceptio { }
static void locate_results2(String command, Color clr) throws Exception { }
static void assoc_results(String command, Color clr) throws Exception { }
void help_results(String title, String contents, int x, int y) { }
```

Each function involves instantiating a result window, setting up its position and color, displaying the query results and registering the result window into its group.

3. User Manual

The following presents a brief tutorial on how to use the VHDL-AMS Browser:

3.1 Starting the Browser

- The user can run this browser in both UNIX and Windows environment.
  In UNIX, the user has to set up the X-Windows environment by command:
  setenv DISPLAY "workstation":0.0
  where workstation is the name of the machine currently being used.

- Compile source code: Gui.java by command: *javac Gui.java*

- Invoke the Browser by command: *java Gui*

A window with title "VHDL Design Browser" pops up in the top-left corner of the screen. There are three parts in this main window: on the top is the menus; in the middle is a list showing "Loaded Files(.vhdl)"; at the bottom is a text area for logging all the browsing actions.

There are six menus in the menu bar: **File, List, View, Locate, Associations** and **Help**. Each has its own menu items. At the start, there are only two menus: **File and Help** enabled, which leads the user either to load the VHDL design by clicking on "Load File(.vhdl)" on **File**, or to terminate the browser by clicking on "Exit" on **File**, or to view help information about other menus by clicking on **Help** menu items.

3.2 Loading the VHDL Hardware Description

- Click on "Load File(.vhdl)" on **File**.

A file dialog box pops up. The user can go through the whole file system on his computer to find the file he wants to browse. If the file is not ended with ".vhdl" or the VHDL file is already loaded, it is ignored without any further action. Otherwise the whole file is loaded into Prolog parser, the VHDL file name (e.g. *tp_7.vhdl*) shows up in the loaded file list and the whole action is recorded in the browsing log as follows:

```
Load File(tp_7.vhdl)
--- waiting ...
--- succeeded
```

And all the menu items related to the VHDL file are enabled. Thus the user needs to only click on those enabled menu items to do further browsing without clicking on those menu item of no results.

3.3 Browsing

The results windows are positioned according to their query category (see Figure 2). Each result window is offset from the previous one in the same query category horizontally.

The result windows are colored by query functionality. E.g. All query results windows about "Entity" have the same color: List All Entities, View: Show Source Code of Entity, View: Show Source Code and Doc of Entity.

All Browsing actions are recorded in "Browsing Log" as follows:

List All Architectures of Entity(all)
--- waiting...
--- finished

Line 1 is the query, the same as the title of the result window, line 2 shows up when the query is submitted to the Prolog search engine, line 3 shows up when the result window pops up. It's better to wait till line 3 shows up before issuing the next query because it takes time to process the current query. If line 3 doesn't show up for a long time, it is likely that there was an error on the Prolog side.

There is an item "CloseAllSubWindows" on each menu or submenu. Clicking on it will close all the results windows triggered by items on the menu it belongs to.

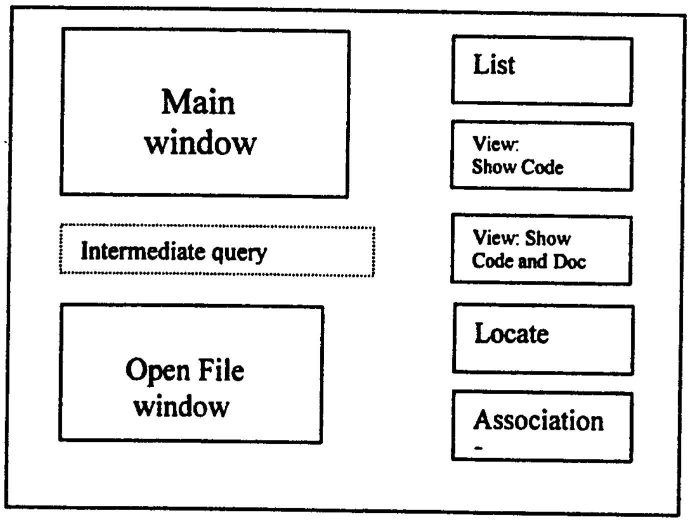


Figure 2: The layout of Results windows

The following sample queries are categorized as follows:

- **Open File**
  - Double-click on the item in the "Loaded File(.vhdl)" list (e.g. tp_7.vhdl).
  - A result window with title "Open File – tp_7.vhdl" pops up.
  - File contents in the text area is not editable.

- **List ...**

- List All Entities
  - Click on item "Entities" on menu **List**.
  - A result window with title "List All Entities" pops up.

- List All Architectures
  - Click on item "Architectures" on menu **List.**
  - An intermediate query window with title " List: architectures" pops up.
  - All associated entities are shown in the drop-down list. User can choose any of them (e.g. blocks) to list all architectures of this specific entity or choose the default one "all" to list all the architectures.
  - Click on "Ok" button to pop up :
    a result window with title "List All Architectures of Entity (blocks)" or
    a result window with title "List all Architectures of Entity (all)".

■ **View: Show Source Code of ...**

- View: Show Source Code of Entity (...)
  - Click on item "Entity" on sub menu **View: Show Source Code of.**
  - An intermediate query window with title "View: show source code of entity" pops up.
  - All entities are shown in the drop-down list. User can choose any of them (e.g. blocks) to show source code of this specific entity.
  - Click on "Ok" button to pop up a result window with title "View Show Source Code of Entity (blocks)".

- View: Show Source Code of Architecture (...)
  - Click on item "Architecture" on sub menu **View: Show Source Code of.**
  - An intermediate query window with title "View: show source code of architecture" pops up.
  - All architectures are shown in a drop-down list and all associated entities are shown in another drop-down list, user need choose one architecture (e.g. blocks) and its associated entity (e.g. blocks or default to be "all") to show source code of this specific architecture(e.g. blocks).
  - Click on "Ok" button to pop up a result window with title "View Show Source Code of Architecture (blocks) with Entity (blocks)".

- Double Click on items listed on "List ..." results windows
  - A result window with title "View: Show Source Code of ..." pops up to automatically show the source code of this item.

■ **View: Show Source Code and Doc of ...**

Browsing process is the same as query "View: Show Source Code of ...".

■ **Locate ...**

- Locate UserDefinedDataTypeDefn of DataType(…).
    - Click on item "UserDefinedDataTypeDefn" on menu **Locate.**
    - An intermediate query window with title:
      "Locate: UserDefinedDataTypeDefn" pops up.
    - All UserDefinedDataType are shown in the drop-down list. User can choose any of them (e.g. var_array) to locate user-defined data type definition of this specific data type or choose the default one "all" to locate definitions of all the user-defined data types.
    - Click on "Ok" button to pop up a result window with title:
      "Locate UserDefinedDataTypeDefn of DataType(var_array)"
      or a result window with title
      "Locate UserDefinedDataTypeDefn of DataType(all)".

- Locate Variable (…) with DataType (…)
    - Click on item "Variable" on menu **Locate.**
    - An intermediate query window with title "Locate: Variable" pops up.
    - All variables are shown in a drop-down list and all associated datatypes are shown in another drop-down list, user need choose one variable (e.g. v0 or default to be all) and its associated datatype (e.g. integer or default to be "all") to locate a specific variable (e.g. v0) of a specific datatype(e.g. integer), or locate all variables of a specific datatype(e.g. integer),
      or locate a specific variable(e.g. v0) of any data type, or locate all the variables of any datatype.
    - Click on "Ok" button to pop up
      a result window with title "Locate Variable(v0) with DataType(integer)", or a result window with title "Locate Variable(all) with DataType(integer)", or a result window with title "Locate Variable(v0) with DataType(all)", or a result window with title "Locate Variable(all) with DataType(all)".

■ **Associate …**

- Associate ComponentHierarchy with top-level Entity (…)
    - Click on item "ComponentHierarchy" on menu **Associations.**
    - An intermediate query window with title:
      "Associations: ComponentHierarchy" pops up.
    - All entities are shown in a drop-down list. User can choose any of them (e.g. blocks or default to be all) to show the component hierarchy of this specific entity or show all the component hierarchy.
    - Click on "Ok" button to pop up a set of results windows with title:
      "Associate ComponentHierarchy with top-level Entity(blocks) or
      "Associate ComponentHierarchy with top-level Entity(all),
      each result window is associated with an architecture of an entity

- Associate X (…) to be subcomponent of Y (…)
    - Click on item "X_subcomponent_Y" on menu **Associations.**
    - An intermediate query window with title

"Associations: X_subcomponent_Y" pops up
• All entities are shown in two drop-down lists. User can choose any of them (e.g. X to be "vbm1" or default to be "all", Y to be "vbm0" or default to be "all") to check if X (vbm1) is a subcomponent of Y (vbm0), or find all component associations with X (vbm1) as the subcomponent, or find all subcompents of Y (vbm0) or find all possible X_subcomponent_Y associations
• Click on "Ok" button to pop up a set of results windows with title "Associate X(vbm1) to be subcomponent of Y(vbm0)" or
"Associate X(vbm1) to be subcomponent of Y(all)" or
"Associate X(all) to be subcomponent of Y(vbm0)" or
"Associate X(all) to be subcomponent of Y(all)" , or
each result window is associated with a X_subcomponent_Y association

■ **Help ...**
- Help: ListQueries
  • Click on item "ListQueries" on menu **Help**
  • A window with title "Help: ListQueries" pops up in the position where all "List ..." results windows are located to describe "List ..." queries.

3.4 Terminating the Browser

Click on item "Exit" on menu **File** to terminate and exit the Browser.

4. Conclusions and Future Enhancement

The primary advantage of this Java-based GUI is its portability. Because Java is a platform independent language, this VHDL Browser can be run in both UNIX and Windows environments.

The GUI is "user-friend" and guides the user instead of letting the user click around blindly. User can easily find out the customized queries by going through the menu bar. And only those menu items that are meaningful in a given context. Result windows are organized according to their group and are colored and positioned sutably to show similarities and differences. The result windows belonging to one group can all be closed at once.

In future, the GUI can be enhanced with editing functionality and more tightly integrated with the VHDL-AMS source files.

__TASK 2__

VHDL-AMS to SPICE Translator

Principal Investigator: Prof. Mohammed Ismail

Institution: Ohio State University, Columbus, Ohio

Period of Performance: August 19, 1997 to March 18, 1998

Task: The contractor shall develop and examine a parser which will convert a VHDL-AMS description to a Spice netlist. This parser will be used with the CERC's development of a Mechatronic Design Environment for the generation of a circuit netlist given the equivalent models at their structural and behavioral levels of abstraction.

The Implementation of A VHDL-AMS to SPICE Converter

Shenggao Li, Brian Okoon, Mona Hella, Mohammed Ismail
Analog VLSI Lab, The Ohio State University

Abstract--The implementation of a VHDL-AMS to SPICE converter (Vhdl2Spice) to be inserted in a complete CAD environment [1] is described. The Vhdl2Spice generates SPICE netlist by tracing down the VHDL IIR parse tree available from the already existing VHDL analyzer. Corresponding to each VHDL design unit, the output of the converter is represented as a subcircuit in SPICE. By doing so, the hierarchical characteristic of VHDL is retained in the SPICE representation; hence future extension to both programs can be parallel. The Vhdl2Spice converter is also a demonstration of the extensibility of the AIRE [2] for a complete integration of VHDL-AMS with other CAD tools.

Keywords: Mixed signal, VHDL-AMS, SPICE, EDA, Language translation

I. Introduction

Integrated circuit technologies have now reached an era where mixed analog/digital signal processing systems in a single chip are in a growing demand [3]. In addition, as the feature size of semiconductor components is scaled down to submicron level, a digital system is likely to exhibit analog behavior such as transmission line effect at high frequency rate [4]. To facilitate the design and simulation of mixed signal processing systems, and precisely model the functionality of a digital system with possible detrimental analog behaviors, a unified EDA medium that can handle both analog and digital signals is thus necessary [4-7]. Fortunately, several efforts have been made to create an analog extension to the essentially digital VHDL hardware description language (IEEE-1076) [7]. Among them is the new IEEE 1076.1 standard (VHDL-AMS), which supports the description and simulation of both continuous and discrete systems [8].

The great flexibility of VHDL allows the integrated design automation including description, simulation, synthesis, formal verification, and testing [4]. While this is true for digital systems, the design automation for analog and mixed-signal systems is still a lasting goal among researchers. Among other things, the complexity of analog signals and the diversity of analog specifications hinder the all-level automation of analog and mixed-signal circuits [9]. In spite of the difficulties, the new extended hardware description language, with its capability in high level behavioral abstraction of a complex system, does show its advantages, namely the high efficiency in describing and simulating complicated mixed signal systems, and the capability to extend the functions such as fault modeling [5], reliability analysis [6], statistical simulation and circuit optimization [3] for the existing VHDL-AMS environment.

As it is the case with analog systems, simulation with SPICE-like simulators is widely used today due to its accuracy and less abstraction. While efforts are taken to enhance the capability of the VHDL-AMS, it is necessary to provide a connection between VHDL-AMS and SPICE. With the connection, design and simulation using either program can be transferred to and verified by the other, or the abstraction of a system at one level can be simulated in one program and then converted to another level of abstraction to be simulated in the other program. Currently, there are already programs that translate SPICE description to VHDL-AMS description [10]. This paper will introduce the implementation of the opposite, namely the translation from VHDL-AMS to SPICE.

A program (Vhdl2Spice) that converts a VHDL-AMS description to SPICE netlist is developed and embedded in an already existing standard VHDL analyzer [1]. The standard VHDL analyzer, which is one of the implementations of AIRE [2], provides an extensible, object-oriented, open intermediate representation (IIR) definition for VHDL. Basically, the analyzer implements the early stages (lexical processing, parsing action, and semantic analysis) of a VHDL compiler, and the resulting abstract syntax tree (AST) from the analyzer is in the form of IIR (In-memory Intermediate Representation). The analyzer is implemented using object-oriented programming. For each VHDL elements (terminals or non-terminals in the BNF of the language), there is a corresponding IIR class abstraction. Fig1-a shows the IIR class hierarchy implementation in the VHDL analyzer. IIRBase class contains the AIRE predefined public methods. IIRScram class is a kind of user specific class in the VHDL analyzer, which is used for VHDL regeneration and other purposes. The IIR class is the abstract representation of an IIR and also the public interface for deriving other IIR nodes. The IIR class hierarchy constitutes the whole AIRE specification. The object-oriented implementation of the AIRE makes it easy for further extension to the VHDL class hierarchy. As shown in Fig1-b, an IIRSpice class is inserted in the existing class hierarchy to implement the Vhdl2Spice function.

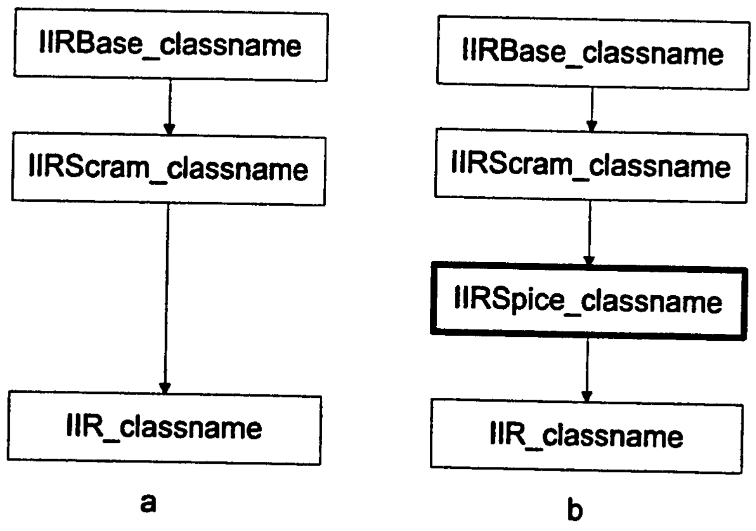


Fig1. a) Implementation of IIR class
b) Extending IIR class for Vhdl2Spice converter

Since the VHDL analyzer has already performed the compiling actions for a VHDL description, our efforts will focus on the analysis of the resulted parse tree. The derivation of the IIRSpice class from the existing class hierarchy makes life easier when implementing the Vhdl2Spice converter. Due to the flexibility of VHDL, not every IIR element defined in AIRE has it counterpart in SPICE grammar, so it is not necessary to write an IIRSpice class for every IIR class. For each implemented IIRSpice class, a method named _publish_spice() is defined to keep tracking of it sub-elements (recursively if necessary). By tracing down the IIR parse tree, the converter will be able to translate the elements in a VHDL design unit to SPICE netlist.

In the following sections, section II discusses the conversion of a VHDL component to a SPICE subcircuit. Section III describes the implementation details of the converter. Section IV will demonstrate the converting result within the capability of the current VHDL analyzer implementation. Since the VHDL-AMS and the available VHDL analyzer are still undergoing changes and improvement, our demonstration will be restricted within the available IIR implementation. Finally, we draw some conclusions on the implementation of the Vhdl2Spice converter.

II. Subcircuit

We know that for SPICE simulation, a complex circuit is composed of some fundamental SPICE elements, such as resistor, capacitor, inductor, diode, mosfet, bjt, and etc. Our solution to the Vhdl2Spice converter requires that a standard component library that contains these fundamental SPICE elements be built in VHDL-AMS. After that, the basic idea for the Vhdl2Spice converter is to convert a complex VHDL component to a SPICE subcircuit. The declaration of a subcircuit abides by SPICE convention. For instance, a subcircuit may contain only the fundamental building elements, or contain other subcircuits.

In its simplest form, the description of a component in VHDL consists of an interface specification (Entity declaration) and an architectural specification (Architecture declaration). It is possible, however, that several architectural descriptions may exist for a given interface specification. One may use different architectural descriptions for a given type of circuit in the following situation [11]:

- Different simulation levels (behavioral, dataflow or structural) of abstraction for a specific hardware description;
- Different technologies such as Bipolar or CMOS may be employed for a generic design;
- In the structural level, many structures are possible for one type of component such as an operational amplifier, depending on design methodology.

The above situations, among others, require us to observe some naming when translating a VHDL hardware description to a SPICE description. For example, the name of a SPICE subcircuit will adopt the form of ComponentName_Identifier, where ComponentName and Identifier are those as shown in the following VHDL description. Identifier is used to identify multiple implementations of a component with the same interface specification.

```
ENTITY ComponentName IS          --interface specification
   PORT (...);                      --input and output ports
   GENERIC (...);                   --physical and other parameters
END ComponentName;
```

```
ARCHITECTURE Identifier OF ComponentName IS   --architectural specification
   --declarations
BEGIN
   --statements part
      --specification of the functionality of the component in terms of its input lines
      --and influenced by physical and other parameters
END Identifier;
```

The input and output ports of a component, which are defined in the PORT clause, correspond to the external nodes of a subcircuit. The physical or design parameters of the component, as specified in the GENERIC clause, will be put after the external nodes in the .SUBCKT statement. The difference between nodes and parameters in the .SUBCKT statement is that parameters are always initialized. If a parameter is not initialized in the GENERIC clause, it will be initialized to zero when translated to its SPICE counterpart. The external nodes and physical parameters of a subcircuit are formal nodes and formal parameters. When a subcircuit is referenced, the actual nodes and parameters are bound to formal nodes and parameters. These actual node names and parameters are available from the component instantiation statements within the architecture statement part. The binding of actual nodes to formal nodes has to follow the *positional association* method. On the other hand, since default parameters are accepted in both VHDL and SPICE, it is our benefit to adopt *keyword association* when binding the actual parameters to the formal parameters - that is, the name of the formal parameter to which an actual parameter is to be bound is specified with the actual parameter [12]. The advantage of *keyword association* is that we don't need to care about the order of the actual parameters. This is actually a convenient way when describing analog circuits, for which sometimes there are a long list of parameters but only a few has to be specified in a design. Fortunately, both VHDL and SPICE program accept *keyword parameters* binding, and t

Having clarified the above ideas, we are able to start our converter design in converting a VHDL hardware description to a SPICE subcircuit.

III. Vhdl2Spice Implementation

The implementation deals will the VHDL intermediate representation (IIR) as mentioned in section I. The analysis will start from the DesignFileList, which may contain as many as VHDL design files. Each design file, as we are familiar with, can have one or more library units such as configuration declaration, package declaration, package body declaration, entity declaration, and architecture declaration (See Fig2).

The entity declaration and architecture declaration together, specify the external interface characteristics of a subcircuit, which include the subcircuit identifier, external nodes, and formal input parameters with default value. The internal netlist of the subcircuit is determined by the architecture declaration. Within the architecture declaration, the component declarations, configuration specifications in the architecture declarative part, and the component instantiation in the architecture statement part are our interests. These descriptions provide information about components that form the component under analysis. They will be translated to subcircuit references during the conversion. There are, however, possibly behavioral descriptions in the architecture declaration, which are more flexible and complicated, and is considered difficult for analysis and conversion so far. At this stage we will not consider behavioral descriptions. For more complicated VHDL application, the binding of a component instantiation to an actual component might be done outside the architecture by using configuration declaration, which just need extra effort during the conversion analysis.

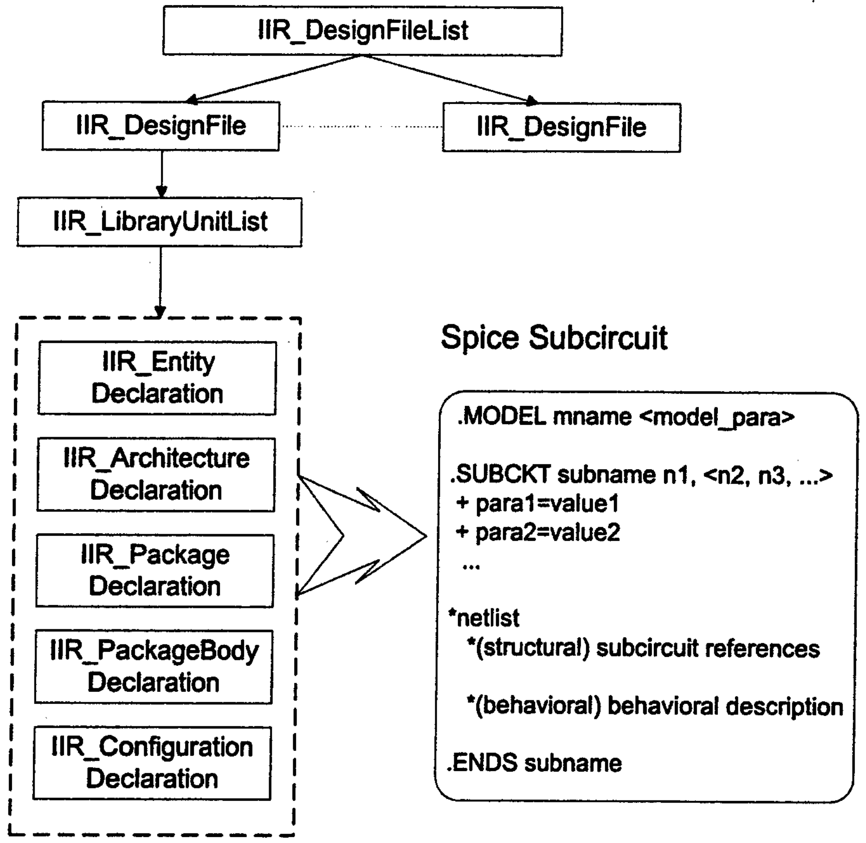

IIR_DesignFileList
IIR_DesignFile
IIR_DesignFile
IIR_LibraryUnitList
IIR_Entity Declaration
IIR_Architecture Declaration
IIR_Package Declaration
IIR_PackageBody Declaration
IIR_Configuration Declaration
Spice Subcircuit
.MODEL mname <model_para>
.SUBCKT subname n1, <n2, n3, ...>
+ para1=value1
+ para2=value2
...
*netlist
*(structural) subcircuit references
*(behavioral) behavioral description
.ENDS subname

Fig2. Implementation of the Vhdl2Spice converter

Furthermore, some of the common parameters, subprograms or components of a VHDL hardware description may be grouped in a package declaration, which is made visible by the USE clause to an entity, architecture or configuration declaration. A package declaration will be converted to SPICE model using the .MODEL device as shown in Fig2. Again, a package may contain function or procedure declarations. They are not included in this stage of work.

Thus far, we should have noticed that, for components defined in the standard library as discussed in section II, we only need to translate their VHDL description to the fundamental component statement. To identify fundamental components from complex components, the entity name of a fundamental component should be reserved and made available for the _publish_spice() analyzer in a set of fundamental component identifiers.

A method is developed exclusively to process the fundamental components, as long as one is identified to be in the fundamental component set.

It is worth to mention that we don't want to convert the test bench to a subcircuit. As a consideration, we might use the testbench identifier to recognize a test bench, or a command line argument can be given during conversion to identify the intended test bench. To abide by the VHDL convention, we choose the later.

IV. Conversion Example

Currently, the Vhdl2Spice converter is implemented in gnu C++ under Linux environment. The IIRSpice class derivation, code modification and program configuration are made according to the instruction given in [2]. Specifically, a "-publish-spice" command line argument is given to the VHDL analyzer to run the Vhdl2Spice conversion. One or more VHDL design files can be given to the Vdhl2Spice converter. After conversion, the output will be stored in files with extension name of ".cir" according to SPICE convention, or the output can be directed to screen if "-no-file-output" option is given in the command line parameters. So far, our test files are still in digital domain due to the limited support of the VHDL analyzer for analog and mixed signal, which should be enough to show how the program works. Once the standard VHDL analyzer has the full capability to process VHDL-AMS description, it will not be difficult to extend the ability of the converter. As a simple example, let's look at the following VHDL description of a non-inverting buffer (Fig3). The buffer is composed of two CMOS inverters, and the primary parameters for the buffer are the width and length for each transistor as shown in the circuit. For simulation purpose, the buffer is described in the following VHDL (93) code.

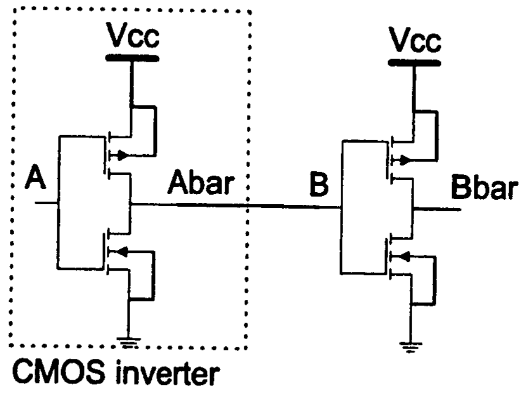


Fig3. A non-inverting Buffer

```
--VHDL description of a CMOS Switch
PACKAGE Pmosfet IS
   constant model: string :="PMOS";
   constant LEVEL: integer :=3;
   constant PHI: real :=0.700000;
   constant TOX: real :=2.9600E-08;
   ...
```

```
 END Pmosfet;

 PACKAGE Nmosfet IS
    constant model: string :="NMOS";
    constant LEVEL: integer :=3;
    constant PHI: real :=0.700000;
    constant TOX: real :=2.9600E-08;
     ...
 END Nmosfet;

 ENTITY mosfet IS
           GENERIC (W: real := 6.0e-6; L: real :=1.0e-6);
           PORT (D, G, S, B: bit);
 END mosfet;

 USE WORK.Pmosfet.ALL;
 ARCHITECTURE P of mosfet IS
 BEGIN
    ...
 END P;

 USE WORK.Nmosfet.ALL;
 ARCHITECTURE N of mosfet IS
 BEGIN
    ...
 END N;

 ENTITY CmosInverter IS
          GENERIC (Wp, Lp, Wn, Ln: real);
          PORT (A: IN BIT; Abar: OUT BIT; Vdd, Vss: In BIT);
 END CmosInverter;

 ARCHITECTURE ex1 OF CmosInverter IS
         COMPONENT mosfet
             GENERIC (W, L: real);
             PORT (D, G, S, B: bit);
         END COMPONENT;
         for m1: mosfet USE ENTITY WORK.mosfet(P);
         for m2: mosfet USE ENTITY WORK.mosfet(N);
BEGIN
        m1: mosfet GENERIC MAP (Wp, Lp) PORT MAP ( Abar, A, Vdd, Vdd);
        m2: mosfet GENERIC MAP (Wn, Ln) PORT MAP ( Abar, A, Vss, Vss);
END ex1;

ENTITY NonInvertingBuffer IS
         GENERIC (Wp1, Lp1, Wn1, Ln1, Wp2, Lp2, Wn2, Ln2: real );
         PORT (Vin: IN BIT; Vout: OUT BIT; Vdd, Vss: In BIT);
END NonInvertingBuffer;

ARCHITECTURE ex1 OF NonInvertingBuffer IS
        COMPONENT CmosInverter
            GENERIC (Wp, Lp, Wn, Ln: real);
            PORT (A, Abar, Vdd, Vss: bit);
        END COMPONENT;
        SIGNAL Vout1: BIT;
        for inv1, inv2: CmosInverter USE ENTITY WORK.CmosInverter(ex1);
BEGIN
        Inv1: CmosInverter GENERIC MAP (Wp1, Lp1, Wn1, Ln1)
                           PORT MAP (Vin, Vout1, Vdd, Vss);
        Inv2: CmosInverter GENERIC MAP (Wp2, Lp2, Wn2, Ln2)
                           PORT MAP (Vout1, Vout, Vdd, Vss);
END ex1;

ENTITY testbench IS
END testbench;

ARCHITECTURE one OF testbench IS
        COMPONENT NonInvertingBuffer
            GENERIC (Wp1, Lp1, Wn1, Ln1, Wp2, Lp2, Wn2, Ln2: real);
            PORT (Vin, Vout, Vdd, Vss);
        END COMPONENT;
        SIGNAL Vin, Vdd, Gnd: IN BIT;
        SIGNAL Vout : OUT BIT;
        for ALL: NonInvertingBuffer USE ENTITY WORK.NonInvertingBuffer(ex1)
BEGIN
        buffer1: NonInvertingBuffer
          GENERIC MAP (12.0e-6, 1.0e-6, 6.0e-6, 1.0e-6, 24.0e-6, 1.0e-6, 12.0e-6, 1.0e-6)
          PORT MAP (Vin, Vout, Vdd, Gnd);
        -- Analysis Action
END one;
```

The corresponding SPICE description looks like:

```
*Spice netlist
.MODEL Pmosfet "PMOS"  level = 3 phi = 0.7000 tox = 2.9600E-08
.MODEL Nmosfet "NMOS"  level = 3 phi = 0.7000 tox = 2.9600E-08

.SUBCKT CmosInverter_ex1 Vin Vout Vdd Vss Wp=0 Lp=0 Wn=0 Ln=0
        m_m1 Vout Vin Vdd Vdd MODEL = Pmosfet  W = Wp L = Lp
        m_m2 Vout Vin Vss Vss MODEL = Nmosfet W = Wn L = Ln
.ENDS CmosSwitch_ex1;

.SUBCKT NonInvertingBuffer_ex1 Vin Vout Vdd Vss
+ Wp1=0 Lp1=0 Wn1=0 Ln1=0 Wp2=0 Lp2=0 Wn2=0 Ln2=0
        X_Inv1 Vin Vout1 Vdd Vss  Wp=Wp1 Lp=Lp1 Wn=Wn1 Ln=Ln1 CmosInverter_ex1
        X_Inv2 Vout1 Vout Vdd Vss Wp=Wp2 Lp=Lp2 Wn=Wn2 Ln=Ln2 CmosInverter_ex1
.ENDS NonInvertingBuffer_ex1;

* ----testbench
X_buffer1 Vin Vout Vdd Gnd
+ Wp1 = 12.0e-6 Lp1 = 1.0e-6 Wn1 = 6.0e-6 Ln1 = 1.0e-6
+ Wp2 = 24.0e-6 Lp2 = 1.0e-6 Wn2 = 12.0e-6 Ln2 = 1.0e-6
+ NonInvertingBuffer_ex1
* ----Analysis Action
```

V. Conclusion

Based on the work we have done so far, some conclusions can be drawn:

- The Vhdl2Spice converter implemented is not restricted to some specific circuits. Using subcircuit, the hierarchical programming characteristic of VHDL is retained in SPICE description. Further extension to both VHDL-AMS and SPICE can be parallel.
- Due to the programming flexibility in VHDL, which is not the case in SPICE, some rules or conventions must be developed and followed when building VHDL-AMS circuit models so as to facilitate the successful conversion.
- The Vhdl2Spice translator implementation is a complementary support for the currently limited capability of VHDL-AMS in terms of analog and mixed signal simulation. The final goal is to implement a seamless integrated EDA package that simulates analog and mixed-signal systems in all levels of abstraction including behavioral, structural, electrical (components), and physical (layout) aspects.

Acknowledgment:

We would like to thank Dr. Robert Ewing and Ms. Maya Rubeiz in Wright Laboratory for supporting this project.

References

[1] "http://www.mtl.com/projects/savant/savpage1.html"; MTL Systems, Inc. (MTL) and the University of Cincinnati: SAVANT – Standard Analyzer of VHDL Application for next-Generation Technology.

[2] "http://www.ececs.uc.edu/~paw/aire/Index.html"; University Of Cincinnati: Advanced Intermediate Representation with Extensibility (AIRE), AIRE is a working group under EIA.

[3] M. Ismail, Terry Fiez: "Analog VLSI: Signal and Information Processing"; McGraw-Hill, c1994.

[4] Alain Vachoux: "Analog and Mixed-Signal Extensions to VHDL"; Analog Integrated Circuits and Signal Processing, V16, June 1998, pp97-112.

[5] A. J. Perkins, M. Zwolinski, C. D. Chalk and B. R. Wilkins: "Fault Modeling and Simulation Using VHDL-AMS"; Analog Integrated Circuits and Signal Processing, V16, June 1998, pp53-67.

[6] Serag M. Gadelrab, James A Barby: "Creative Methods of Leveraging VHDL-AMS-like Analog-HDL Environments. Case Study: Simulation of Circuit Reliability"; Analog Integrated Circuits and Signal Processing, V16, June 1998, pp69-83.

[7] Lun Ye, H. W. Carter: "Analog Behavior Modeling and Processing Using AnaVHDL"; Analog Integrated Circuits and Signal Processing, V16, June 1998, pp85-95.

[8] "Definition of Analog Extensions to IEEE Standard VHDL (1076.1 draft)"; IEEE, July 1, 1997.

[9] M. Ismail, Jose Franca: " Introduction to analog VLSI design automation"; Kluwer Academic Publishers, c1990.

[10] S. R. Mayiladuthurai AND H.W. Carter: "Translating SPICE Models to VHDL-AMS"; VHDL International Users Forum, Sept. 1996.

[11] Zainalabedin Navabi: "VHDL—Analysis and Modeling of Digital Systems (2nd edition)", McGraw-Hill, 1998.

[12] Robert W, Sebesta: "Concepts of Programming Languages (3rd edition)", Addison-Wesley, 1996.

__TASK 3__

Component Library Development for Mechatronic Design Environment

Principal Investigator: Prof. Frank Scarpino

Institution: University of Dayton, Dayton, Ohio

Period of Performance: August 19, 1997 to December 1, 1997

The final report for this task is unavailable. It will be included within this report when available.

TASK 4

Macro Modeling Tool for VHDL-AMS

Principal Investigator:	Profs. Joann Degroat and Steve Bibyk
Institution:	Ohio State University, Columbus, Ohio
Period of Performance:	March 24, 1998 to September 1, 2001
Task:	The contractor shall develop, evaluate, and distribute a macro modeling tool for VHDL-AMS. This macro modeling tool will build upon the Digital Model Generator developed at Ohio State under the RASSSP program. The objective of this task is to automate the design, development and implementation of mixed signal circuits.

System-on-a-Chip Design Methodologies and Issues for Transducer-to-Pico-Network Applications

John Sheridan Fisher, Robert Henz, Deepika Devarajan, Jason Abele, and Steven B. Bibyk, Information Electronics Research Group, The Ohio State University Department of Electrical Engineering, Columbus, OH 43210, ie@ee.eng.ohio-state.edu

Abstract

The unique phenomenology of transducers complicates designing the interface of the transducer-to-pico-network protocol. This paper will explore the methodologies and issues involved in the design flow of a mixed-signal microchip through a sensor-to-pico-network design example that has been fabricated in the AMI $0.5\mu m$ process. The design methodologies include a careful decomposition between the digital and analog sections, as well as between the individual analog blocks, which decreases time-to-market and enhances design reuse. The fabricated design includes a $4-20mA$ receiver, a second-order Sigma-Delta Modulator, fourth-order sinc filter, and an SPI block.

I. Introduction

The design of the transducer-to-digital-pico-network interface is a difficult and complex task, primarily because the phenomenology of the transducer is generally not well characterized over electrical and atmospheric (temperature, pressure, humidity, etc.) variations. In contrast, the digital pico-network or LAN protocol is generally well defined in a specification, such as IEEE 1451 [1]. For our discussion, we will assume that a sensor generates an analog electrical signal that we wish to send over a digital network. This topology has a large variety of practical applications where size, power, or weight constraints merit a single chip design solution and include wireless audio acquisition, blimp mounted video surveillance, and remote environment monitoring all via a wireless connection to a self-configuring pico-network.

Our design example is know as the Nautilus chip, which is an integrated microchip solution for receiving a standard $4-20mA$ analog communications protocol, converting the received analog to $16-bit$ digital words, and transmitting the digital words on a standard SPI (Serial Peripheral Interface) pico-network connection. The ADC (analog-to-digital converter) uses a second- order Delta-Sigma A/D topology and is followed by a third-order sinc filter. The SPI core transmits the 16-bit digital words across a standardized, fully configurable SPI serial interface as 16-bit/8-bit data. In addition, the SPI core can receive data for the ADC, such as filter coefficients. The chip was fabricated through MOSIS on AMI's $0.5\mu m$ process.

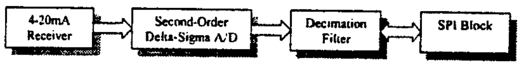


Fig. 1. Nautilus Chip Block Diagram.

II. Design Flow

This paper will follow the methodologies and issues involved in the design flow shown in Figure 2. At the top, we begin with a behavior model, such as VHDL-AMS, of the top-level system. This model is interactively simulated and developed. Obviously, the most important part of this model will be accurately capturing the interfaces between the blocks and then ensuring the actual produced design meet these interface specifications. In addition, this model should lend directly to initiating the design of the VHDL for the digital sections.

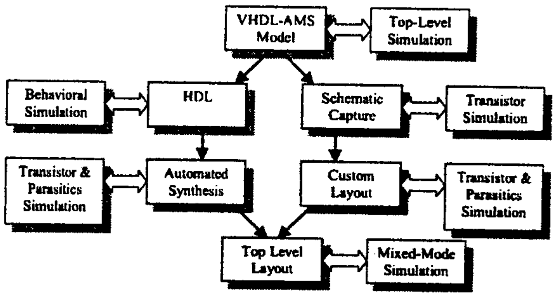


Fig. 2. Integrated Mixed-Signal Design Flow Diagram.

Once the top-level model had been solidified, the mixed-signal design can be broken into the analog parts and the digital parts. Both the digital section and the analog section are also interactively simulated and developed. The digital section is developed with behavioral models in VHDL, while the analog section is developed at transistor-level with schematics. Once these designs are solidified, the digital design goes to layout through an automated synthesis path. In contrast, the analog design goes to the physical layer through full custom layout, which can then be extracted with parasitic passives devices for re-simulation. Finally, the synthesized layout is inserted into the full custom layout. This final top-level physical representation of a complete system is usually too complex to extract and to simulate at the transistor level, so special mixed-mode simulation tools exist to simulate the digital designs with behavior model in tandem with simulating the analog sections with approximations of the full transistors models. These mixed mode simulations are primarily intended to prove functionality, rather than detailed performance.

In our design example, the digital blocks of the mixed-signal chip were designed as a reusable core to aid in future projects. These digital designs were initially developed and tested in behavioral VHDL using Mentor Graphics Tools. This description was then ported to the behavioral VHDL subset supported by the Alliance Tools in order to use their free logic synthesis and automatic place- and-route tools. The analog section design, simulation, and layout, as well as top-level layout, were done in the Cadence Toolset.

III. Analog First

The first step in a mixed-signal design is to fix the resolution that is to be acquired from a sensor or sent to an actuator, which will constrain the choice of A/D or D/A, respectively. Since the design methodology for a D/A with actuator is similar to the sensor with A/D case, we will only consider the latter. The next step is to specify the interface between the A/D and the sensor signal conditioning electronics. For example, we may specify the input range of our A/D. Then, the full output range of the signal conditioning should be a subset of the input range. This subset must balances a trade-off between keeping the output range wide enough not to waste resolution of the A/D and narrow enough that manufacturing variations in the sensor and analog electronics will not cause a misalignment between the sensor output and the A/D input, and thus a "dead-zone" in the sensor.

This decoupling of the A/D input and signal conditioning output is essential for moving the project forward more quickly. The design of the A/D can start once this interface has been specified and is independent of the exact choice of sensor or its individual phenomenology. In parallel, a particular sensor can be chosen and characterized without impeding the rest of the design flow. The sensor signal conditioning can begin to take form as soon at the interface is specified and can congeal more fully as the sensor phenomenology is explored.

A. $4-20mA$ Receiver

The analog section of our mixed-signal chip senses an incoming current and converts its magnitude to a digital format. Specifically, we have designed a receiver for a standard $4-20mA$ current loop and a second-order Delta-Sigma A/D to generate a digital pulse density stream of the data on the current loop. We chose the standard $4-20mA$ current loop as our "sensor" data because its electrical signal behaves like sensor data which we can explore for its design methodologies without the overhead of uncertain sensor phenomenology.

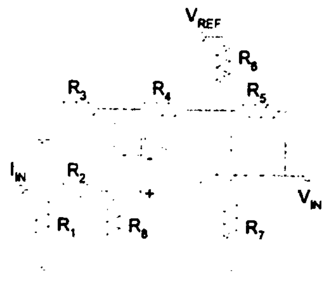


Fig. 3. $4-20mA$ Receiver Schematic.

The $4-20mA$ receiver is a simplified version of a topology developed by Burr- Brown [2] and is shown in Figure 3. The transfer function for the $4-20mA$ receiver is given by Equation 1 and Equation 2. As shown by Equation 1, V_{REF} can be adjusted to center the range of the receiver's output V_{IN} within the A/D's input range. If V_{REF} is made to be tunable, then each individual receiver can be tuned within its assembled $4-20mA$ system to account for inconsistencies in the sensor input or analog block fabrication.

$$V_{IN} = \frac{R_1 R_8}{R_2 + R_1 + R_8} k I_{IN} - \frac{R_5}{R_6} V_{REF}, \quad (1)$$

where

$$k = 1 + \frac{R_4}{R_3} + \frac{R_5}{R_3} + \frac{R_5}{R_7}(1 + \frac{R_4}{R_3}) + \frac{R_5}{R_6}(1 + \frac{R_4}{R_3}) \quad (2)$$

B. Second-Order Delta-Sigma Modulator A/D

Our chip's A/D design is a second-order Delta-Sigma Modulator (DSM) A/D whose output is to be decimated to $16-bit$ resolution. A DSM was chosen because of its high resolution, low die area consumption, and unique tolerance to fabrication process variations. Also, using a DSM forces the use of digital filtering, which can be used to correct a spurious response that might be found in the analog section as the design progressed. If other A/D topologies suffered this problem during design, a separate DSP would have to be added to correct spurious response; this addition would represent a major change in the design specification and seriously impact time-to-market. Our design for the DSM was developed by combining aspects of a third-order DSM by Johns & Martin [3] and a second-order design discussed by Mandyam [4], as well as general topological considerations from Boser [5] and Shenoi [6].

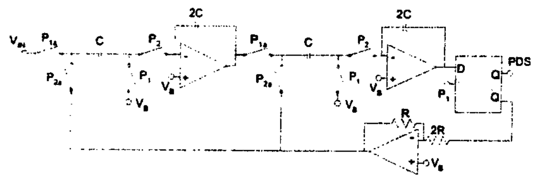


Fig. 4. Second-Order Delta-Sigma Modulator Schematic.

We chose a single-ended design because it is sufficient to explore design methodologies involved without the being encumbered by the design overhead of a fully differential architecture. We use a D-flip-flop as a single bit quantizer, which will guarantee linear quantization. Our DSM is designed with the AC ground at a mid-rail bias voltage of V_B. Thus the feedback voltage is actually midway between a rail and V_B, which minimizes the maximum quantization error. We choose an over-sampling rate (OSR) of 128. This topology yields a theoretical maximum signal-to-noise ratio (SNR_{MAX}) of $100.2dB$ (just over 16.5 bits), which is a $34.6dB$ increase in SNR_{MAX} compared with the same design choices using a first-order modulation scheme. It might then seem natural to use an even higher-order topology, however third- order and higher topologies have diminishing returns for SNR_{MAX} and do not have guaranteed stability [3].

In our design in Figure 4, the switched-capacitor structures each form a delaying non-inverting integrator. The signals $P1$ and $P2$ are a non-overlapping clocking pair, as are $P1a$ and $P2a$. The signal $P1a$ is a slightly delayed version of $P1$, which helps attenuate signal feedthrough [4]. Assuming ideal integration, the signal transfer function (STF) is given in Equation 3, while the noise transfer function (NTF) is given by Equation 4. Of course, the pulse density stream (PDS) must be converted into $16-bit$ words by a decimation filter, which will be discussed in the next section.

$$STF = \frac{z^{-2}}{1 - z^{-1} + z^{-2}} \tag{3}$$

$$NTF = \frac{(1 - z^{-1})^2}{1 - z^{-1} + z^{-2}} \tag{4}$$

IV. Digital Design in Parallel

Also in a parallel design path to the analog blocks, the digital sections can be developed. Once an automated synthesis path from VHDL to layout has been set-up and proved for the project, the digital blocks can be re-designed and re- simulated in VHDL, with features being added and dropped, up until the exact time that all of the blocks need to be finished for top-level integration. In stark contrast, simulation of analog blocks is more complex and takes longer than their digital counterparts. Even small changes in the design during the simulation phase force the designer to re-simulate all of the analog test benches, which may significantly delay the schedule. And once custom analog layout has been started, changes in the design will cause the schedule to suffer an even more serious penalty. Given this insight, it should be apparent that the design of the analog blocks of a system of our topology would dictate the schedule. The analog designs will take the longer to complete, and thus their interfaces and specifications need to be decided first. Then each analog and digital block can be designed in parallel. In fact, as inherent weaknesses are found in the analog blocks during their design, often features can be added to the digital sections to compensate, such as changing a digital filter to flatten a passband that had been skewed by an analog block.

A. Decimation Filter

The decimation filter section of our chip takes the pulse density stream provided by the A/D as an input. It then decimates the clock rate by a factor of 128 and yields a $16-bit$ value as an output. The $16-bit$ output value is finally passed to a buffer which can be read by the SPI unit at any time. To achieve this decimation, we implemented a fourth-order sinc filter based on a design by Hogenauer [7]. The sinc filter was chosen for its inherent implementation stability, requiring only digital adders instead of multipliers and adders.

Our design modified the design example by Hogenauer to take a bit stream input instead of a $16-bit$ input. Our design uses $16-bit$ wide columns of integrators and comb filters to do the decimation. Each integrator column is comprised of 4; $4-bit$ carry-look-ahead adders tied together to create a

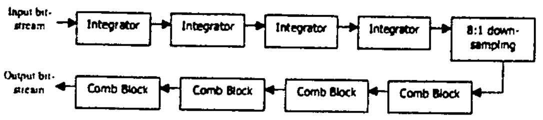


Fig. 5. Decimation Filter Block Diagram.

16 – *bit* adder which takes the previous value (stored in a register) and adds the new input. Similarly the comb-filter columns are made of four, 4-bit comb filters with a differential delay of one. The pulse density stream (our input) is placed on the least significant bit of the first column of integrators.

B. SPI Block

The SPI is essentially a three-wire serial bus for eight or sixteen bit data transfer applications. The three wires carry information between devices connected to the bus. Each device on the bus acts simultaneously as a transmitter and receiver. Two of the three lines transfer data (one line for each direction) and the third is a serial clock. Some devices may be only transmitters while others only receivers [8]. Generally, a device that transmits usually possesses the capability to receive data also.

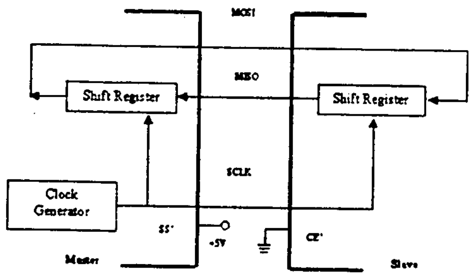


Fig. 6. SPI Block Diagram.

All lines on the SPI bus are unidirectional: The signal on the clock line (SCLK) is generated by the master and is primarily used to synchronize data transfer. The master-out, slave-in (MOSI) line carries data from the master to the slave and the master-in, slave-out (MISO) line carries data from the slave to the master [9], [10].

V. Climbing the Design Hierarchy

A. ESD Protection

Another separate parallel design path is the collection of cells for the chip's pad frame, which typically includes the ESD protection. Almost without exception, the ESD protection cells must be designed through trail-and-error, then tested and documented. These cells rarely conform to the layout design rules and cannot be simulated, since ESD is not accurately modeled. As was the case in our design example, often the foundry will provide these cells to encourage designers to use their process, and also since they will be the most knowledgeable about their process and how design rules may be violated and still produce reliable silicon.

A consequence of using irregular cells that do not pass the design rule check (DRC) is that, although an automatic comparison of the top-level layout to top- level schematic should be possible, it will not be possible to fully check the top-level design for design rule violations (DRVs). The best alternative is to run the DRC on the padframe alone and then on the actual system layout (with routing to the pad locations) alone, and finally on the combined top-level chip. Assuming that the system layout has no DRVs, then the DRC on the padframe and the complete microchip should have the same number of DRVs. While this method is not without fail, most other options, such re-writing the DRC database, will only mask the DRVs, which will in fact still exist; they are only known not to be an issue because the ESD cells have already been fabricated and tested.

B. VHDL IP

In addition to getting ESD structures from outside sources, the VHDL for some of these digital sections may be purchased or acquired for free from various repositories of digital IP, including foundries, design houses, microchip manufacturers, and universities. Often microchip manufacturers will give away VHDL IP to allow designers to create an FPGA/ASIC that will interface with their microchips, which is often true with network protocol chips. Thus, the VHDL for the entire network protocol section may be readily available for synthesis, thus expediting design.

C. Speed Sensitive Digital Sections

Before any digital section is synthesized those sections where speed is of concern should be identified for special treatment. If a regularly used composite function is causing the speed concern, a custom leaf-cell can be created to perform the function then included in the synthesis library. However, often specific blocks or sub-blocks will be causing the speed concern. These blocks should be treated and laid out like analog blocks, although the standard digital block pitch should be maintained to facilitate their interface with the rest of the digital section.

D. Noise at Top Level Integration

Once the digital sections have been synthesized, they will have to be spliced together with the analog sections in the top-level layout. Generally, this is done in the same tools as the custom analog layout. The most notable issues in this co-location stem from a variety of noise injection sources, to which some analog sections will be sensitive. These sections should be carefully isolated from the noise sources. One source of noise injection found in bulk processes is substrate noise. Transistors, and especially those having signals with (digital) secondary higher frequency spurs, can inject high frequency noise into the substrate. In bulk processes, where all NMOS transistors share a common substrate, this noise source can easily perturb reference voltages beyond their required accuracy. Surrounding noisy blocks (or noise-sensitive blocks) with guardrings may lessen the disturbance to (from) nearby blocks. However the guardrings do not completely penetrate the bulk, so noise will still travel under the guardrings, although the impedance seen by noise to nearby blocks is increased. Since noise will travel under the guardrings, carefully developing the microchip's floorplan to keep noise sources and noise-sensitive blocks apart becomes critically important.

Another source of noise is crosstalk between electrical conductors, including the substrate, although usually considered between metal lines. AC signals on one conductor can inject noise into another conductor through the parasitic capacitance between them. Again the most effective method of dealing with this issue is careful floorplanning, although inserting a DC biased (usually grounded) or floating conductor between the two conductors of concern will greatly decrease their parasitic capacitance.

A third noise source that often occurs because of the co-location of analog and digital block in the top-level layout is power supply/ground noise. Often digital blocks are designed with a specified power supply rejection ratio to deal with power fluctuations. However, most data converters and reference voltage generators use ground as their reference point, so ground noise can become a serious issues.

VI. Conclusions

We have discussed design methodologies and issues specific to transducer-to- pico-network applications. Our decomposition of the design has resulted in well- specified individual blocks, which can all be designed in parallel and decrease our time-to-market. In addition, since we now have well specified independent blocks, we can more easily reuse them in future revisions or new designs. If testing of the first spin of a chip reveals that our expectation of the sensor phenomenology was inaccurate, our decomposition allows us to only modify the analog front-end block to re-spin the design. In the same manner, new designs using entirely different sensors can be quickly developed.

We have presented a design example for receiving a standard $4-20mA$ current loop signal and transmitting its $16-bit$ digital representation on an SPI pico- network. Our design example has been used to illuminate the design methodologies and issues presented in this paper. The discussion of the Delta-Sigma Modulator has shown the advantage of creating as much of the design in the digital domain as possible, where signal processing can be designed more quickly than in the analog domain with robust behavior tools and without the issues of fabrication process variation. These chips have been fabricated and are presently being test with results pending soon.

Our design example has shown a methodology for getting analog information from/to a self-configuring network. Already there is a demand for remote sensors to be able to autonomously connect to an established network and post data automatically. The network may also respond with instructions for an actuator at the remote site. The demand for weather station data and (controllable) traffic camera video on the Internet are trivial examples of these sorts of applications. As (remote) wireless applications continue to become more prevalent, the need for integrated systems of this sort will dramatically increase.

References

[1] Kang Lee, "A standard in support of smart transducer networking instrumentation and measurement," *Proceesing of the 17th IEEE IMTC Technical Conference*, pp. 525–528, 2000.

[2] Burr-Brown, "RCV420, Precision 4mA to 20mA Current Loop Receiver," http://focus.ti.com /docs/ prod/ productfolder.jhtml? genericPartNumber=RCV420.

[3] David A. Johns and Ken Martin, *Analog Integrated Circuit Design*, John Wiley & Sons, Inc, New York, 1997.

[4] Bharath Mandyam, "Design issues in i and ii order sigma-delta modulators," M.S. thesis, The Ohio State University, 1999.

[5] B.E. Boser, *Design and Implementation of Oversampled Analog-to-Digital Converters*, Ph.D. thesis, Stanford, 1989.

[6] Kishan Shenoi, *Digital Signal Processing in Telecommunications*, Prentice Hall PTR, New Jersey, 1995.

[7] Eugene B. Hogenauer, "An economical class of digital filters for decimation and interpolation," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, pp. 155–162, Apr. 1981.

[8] ATMEL, "SPI Serial EEPROMs 1M (131,072 x 8) AT25P1024 Preliminary," http://www.atmel.com/ atmel/ acrobat/ doc1082.pdf.

[9] Roger L Stevens, *Serial PIC'n*, Square 1 Electronics, Kelseyville, CA, 1999.

[10] Inc. Microchip Technologies, "PIC16C63A / 65B / 73B / 74B, 8-Bit CMOS Microcontrollers with A/D Converter, 10.0 Synchronous Serial Port (SSP) Module," .

Design Methods for Fabrication of an Analog Network Protocol Chip

Jason Abele and Steven B. Bibyk

Information Electronics Group
Department of Electrical Engineering
The Ohio State University
Columbus, OH 43210
abele.5@osu.edu, bibyk@ee.eng.ohio-state.edu

Abstract

The demands of embedded sensing for miniaturization and low power consumption require the development of integrated mixed signal solutions to bring the wealth of analog sensing to the digital networked world. Through the design and fabrication of a 4-20mA receiver to Serial Peripheral Interface chip, design methods and the space for improvement will be explored.

1. Introduction

As a University chip design group, our primary focus is the development of mixed signal systems and design methods. Our designs target the embedded sensing market where highly integrated System on a Chip (SoC) designs are the desired results. The Nautilus Project, a 4-20mA receiver to Serial Peripheral Interface (SPI), was developed as a proof of concept design to further our embedded sensors to network design goal. A portion of this project included the design of a tool flow that would lend itself to the integration of this entire project into a mixed signal SoC. Our tool flow led to the successful fabrication of the Nautilus chip last summer through a MOSIS submission to the AMI 0.5µm process. Experience has shown pitfalls and triumphs of these methods that can be used to shape future design strategies.

2. Solid tool flow

A major part of successful fabrication is developing and understanding the tool flow that will be used. The current state of computer-based tools tends to divide the world into two parts, digital and analog. The challenge of mixed signal design is finding a way to bridge this divide in the creation of a single system. In figure 2.1 a tool flow is presented that offers a reasonable approach to design, while looking forward to future improvements in computer software.

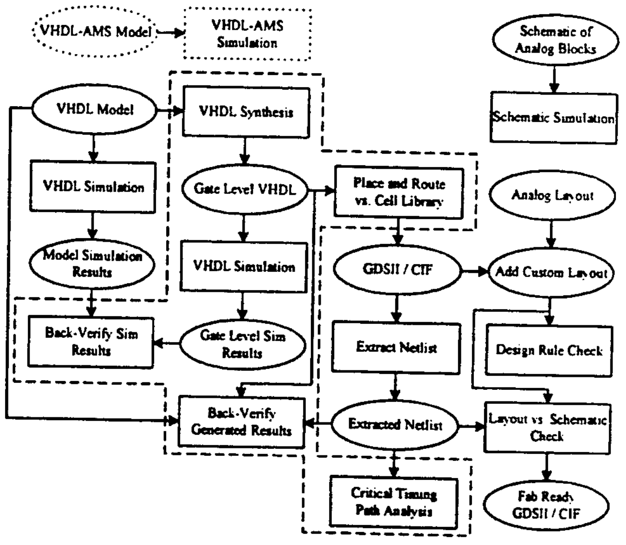


Figure 2.1

On the digital side, tools exist to automate the transition from hardware description language (HDL) to fabrication. VHDL offers a convenient platform to model the system at a high-level, verifying correct function by simulation. After simulation, a synthesis tool is used to reduce the high-level VHDL model to a gate-level description matching a set of gates in the cell library. This gate-level model can be simulated with the same tests used on the original model and verified that any optimization has not changed the function of the design.

The place and route tools (PAR) use the gate-level description to wire cells from the library together and achieve a layout of the design in Calma Format (GDSII) or Caltech Intermediate Format (CIF). The layout represents the layer masks that will be used to fabricate the design. With the layout complete, an extraction tool can use process dependent information to produce a net-list that models parasitic capacitance and resistance. This net-list model can be used to discover critical timing paths that are the determining factor in maximum operating speed. It can also be verified to match the higher-level models from which it was derived.

The analog side of design currently lacks complete computer generated flow from high-level model to finished layout. Our design strategy captures the analog design at the schematic level. Simulation of the schematic model is used to test for correct system behavior. The completed schematic is then used to redevelop the design at the mask level. With the custom layout complete, the design is extracted to produce a net-list. The extracted net-list is then run through the same test simulations as the schematic, to verify that parasitics will not adversely affect the design. Lastly the LVS tool, layout versus schematic, is used to ensure that the manual layout matches the designed schematic.

At the system level, management of the separate design efforts is the key to success. The design teams must be kept aware of their allocations for die space, pins and the specs for interfaces between blocks internal to the chip. Although it is expected that these factors may vary as the design progresses, changes need to be clearly communicated between the design groups. As the blocks are completed, the layouts should be placed in the pad frame and wired into the pads and other blocks as needed. With all the blocks and pads in place, the Design Rule Check (DRC) should be run again and LVS can be used to verify that the blocks have been wired together properly. System level simulations can be run against the final layout; however, most designs will have a level of complexity that precludes detailed system level testing. If the blocks can be simulated against each other at a more abstract level, for example, in an HDL, then a subset of the tests can be run against the final layout to verify expected behaviors from the abstract models. After passing the possible set of system level tests, the chip is ready for fabrication.

3. Existing tool flow

At the outset of the Nautilus project, previous work had established two major portions of the tool flow. The designers knew several packages for the production and simulation of designs in VHDL. The ModelSim VHDL environment was well known from use in coursework and was widely available in the UNIX computing labs. Because the ModelSim license only included the simulation tools, the custom design packages from FPGA manufacturers (Altera and Xilinx) were preferred in the lab courses. None of the tools offered a path to layout for the production of custom silicon. For the production of silicon, two tool suites were available in the computer labs. Magic is preferred in the early coursework for its shallow learning curve and the ability of students to download it for use at home. The advanced classes and research labs prefer the Cadence environment for its robust set of tools to manage the complete design flow from schematic capture to layout, including tools for simulation and verification.

4. The missing link

A goal of the Nautilus project was to fill the gap in our tool flow between HDL and layout. A synthesis tool would automate the transition from a high-level design in VHDL to gate-level before layout. A verification tool was needed to assure functional equivalence between the synthesized gate-level design, layout and the original VHDL model; removing the need for hand verification of simulation results from two separate tools. A cell library and PAR tool would automate the generation of layout, allowing the digital blocks to stay fluid during more of the design cycle. These added degrees of automation would enable digital blocks of much greater complexity to be designed, even on a short time scale.

5. Our tool flow

For the Nautilus Project, the established analog design flow through Cadence and the NCSU Design Kit was left undisturbed [3]. Once the basic system architecture was specified, work began on schematic capture of the design in Cadence. The analog team was given free reign to work through simulation and layout in the Cadence environment. As the analog blocks reached layout, die space and pin allocations were adjusted to meet the needs of the analog blocks.

The digital effort was split, with most of the designers tasked to developing VHDL models of the digital blocks while part of the team focused on integration of the Alliance (© ASIM/LIP6 Université Pierre et Marie Curie) VHDL package into the tool flow. ModelSim was retained as the primary VHDL development and simulation environment due to workstation availability and existing knowledgebase. The Alliance tools were selected because they offered a complete VHDL to mask level tool flow, including cell libraries, and could be freely downloaded to run on Linux. As VHDL blocks passed simulation, they were tested with the Alliance synthesis tool, scmap, to ensure proper syntax and reduction to gate-level VHDL that matched the cell library from Alliance. As the project progressed, the Alliance PAR tool, scr, was used to place and route the cells. This result was then passed to s2r for flattening to GDSII stream for the AMI 0.5μm through MOSIS allowing for improved die size estimates.

With the analog blocks complete, the VHDL model synthesized and the MOSIS submission date approaching, the various blocks were ready to be merged into a chip submission. The analog blocks were added to the assembled pad ring, wired into place and surrounded with the necessary isolation structures. The PAR tools were used to fit the digital blocks in the remaining payload area. The digital blocks were streamed into Cadence and added to the pad ring and analog blocks. After wiring the digital blocks into place, a final pass of the DRC and LVS tools was used to ensure proper wiring and compliance with the process rules. Finally, the complete design was streamed out for delivery to MOSIS.

6. Alliance issues

In the Nautilus project we ran into several problems with the Alliance tools. Understanding the VHDL subset supported by Alliance was the most troubling problem. Although the subset is well documented, our designers who were used to ModelSim found the subset to be cramped [1]. A certain portion of this boiled down to the need to understand the differences between the modeling and synthesis constructs of VHDL. The second most troubling issue was the lack of support for routing in more than two metal layers. Thus we did not take advantage of the metal3 layer in our process run and our design was not as compact as we wished. Finally, when we were preparing this design, the functional extraction tool, yagle, and the static timing analysis tool, tas, were not available. Without tas, we were unable to adequately gauge our critical timing path to allow better design optimization. Our design could only be back verified by simulation results instead of direct comparison with the VHDL models. Having said that, tas and yagle are now both available from Avertec [2].

7. Our experiences

Through fabrication, we learned to adapt quickly to the above known barriers and unforeseen hurdles which appeared during the design process. The largest roadblock was under estimating the effort required to produce a technology file for the Alliance tools. The needs of the technology file are described in the documentation for the tools. However, because Alliance takes a slightly different approach to process independent layout than the MOSIS scmos rules, development and verification of the technology file was slower than anticipated. In order to ease time constraints, we used the known good DRC system in Cadence through the NCSU design kit. Another potential problem was avoided by the flexibility of our digital design flow when it was discovered that our complete design would not fit onto a single tiny chip submission. After carefully dicing our design along functional boundaries, we were able to successfully spread it to multiple chips. With a VHDL based digital tool flow, production of new layout blocks was handled through the place and route design flow with our divided design.

8. Future tools

Looking ahead, there are new tools on the horizon that will advance our design methods even further. We are integrating VHDL-AMS and Verilog-A as a complete system simulation tool in the current respin of the Nautilus and other design projects. These tools will make more complete system tests possible to verify our design before fabrication. Cadence has recently added a complete digital HDL tool chain to the university package that we are investigating to improve the integration of mixed signal design. Analog cell libraries are an area of research that we are tracking in the hope to someday have the automatic place and route capabilities for the analog blocks already enjoyed in our digital flow. With continued research in analog synthesis, it may be possible to bridge that final gap between the high-level representations in a language like VHDL-AMS and the layout that is ready for fabrication.

9. Conclusions

Our methods developed for the Nautilus project have proven successful in producing a first revision of the design. We have the results of our MOSIS submission undergoing thorough testing with very positive preliminary results. The knowledge acquired from the first run is being used to update our methods and design as the Nautilus project updates the design for resubmission later this year.

10. References

[1] Alliance, http://www-asim.lip6.fr/alliance/
[2] Avertec, http://www.avertec.com/
[3] NCSU Design Kit, http://www.ece.ncsu.edu/cadence/CDK.html
[4] VSI Alliance, *VSI System Level Design Model Taxonomy*, Version 1.0, http://www.vsi.org/, October 1998.

TASK 5

Application of Scaleability and Testability in Mechatronic Design Environment

Principal Investigator:	Prof. Janusz Starzyk
Institution:	Ohio University, Athens, Ohio
Period of Performance:	March 24, 1998 to November 30, 1998
Task:	The contractor shall develop, evaluate, and distribute behavioral models for computer simulation of various modules of mechatronic systems like: power controls, power switching, motors, rectifiers, servomechanisms, hydraulic and pneumatic devices, etc. The developed models shall include selection of dominant parameters for identification purposes. The specific application within the Mechatronic design environment will be design of single chip MEM (Microelectromechanical) devices.

The final report for this task is unavailable. It will be included within this report when available.

TASK 6

Parameterized and Distributed Power Regulator Component for Mechatronics

Principal Investigator:	Prof. Marian Kazimierczuk
Institution:	Wright State University, Dayton, Ohio
Period of Performance:	March 24, 1998 to November 30, 1998
Task:	The contractor shall develop a library of precision, parameterized, distributed power regulator components. Thse power regulators will be applicable in the Mechatronic design environment.

The final report for this task is unavailable. It will be included within this report when available.

THE DESIGN OF AN ON-CHIP POWER SUPPLY FOR MICRO-ELECTRO-MECHANICAL SYSTEMS

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Engineering

By

DAVID C. HANNA
B.S., Wright State University, 1999

2001
Wright State University

WRIGHT STATE UNIVERSITY
SCHOOL OF GRADUATE STUDIES

April 16, 2000

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY David C. Hanna ENTITLED The Design of an On-Chip Power Supply for Micro-Electro-Mechanical Systems BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Engineering.

Marian K. Kazimierczuk, Ph.D.
Thesis Director

Fred D. Garber, Ph.D.
Department Chair

Committee on
Final Examination

Marian K. Kazimierczuk, Ph.D.

Raymond E. Siferd, Ph.D.

Fred D. Garber, Ph.D.

Joseph F. Thomas Jr., Ph.D.
Dean, School of Graduate Studies

Abstract

Hanna, David Charles. M.S., Department of Electrical Engineering, Wright State University, 2001. The Design of an On-Chip Power Supply for Micro-Electro-Mechanical Systems.

With the ever-expanding field of micro-electronics, things are now feasibly possible which once were within the realm of science fiction only. One such area of expansion involves the use of Micro-Electro-Mechanical systems (*MEMs*). Their myriad possible uses are now being aggressively explored to further several disciplines within the engineering field. Maybe the most exciting potential use is with respect to micro-biological research. While these devices might use the body as a source of power, on-chip devices presently must rely on discrete power supplies.

This thesis looks at the implementation of an on-chip power supply. Design of the power supply was followed by simulation to test the circuit's proper operation. Designing was broken up into three distinct phases. The first was to develop an operational amplifier (op-amp). The second was in choosing at what power level the power supply should operate at. The last phase was to create and test a dynamic feedback system to compensate for the fast switching nature of *MEMs*.

A power supply was developed that could handle up to 500 mA designed around an output of 2.3 V. With the rapid development of micro-electronics, the necessary technologies should be in place to develop this power supply within the foreseeable future.

Contents

List of Figures

Acknowledgments

I would like to thank both my advisors, Dr. Marian K. Kazimierczuk and Dr. Raymond E. Siferd for their infinite patience while my thesis progressed towards its completion. I would also like to thank Dr. Robert L. Ewing for his support and thoughts. While I did not seek much direction, when it came down to seeking direction/advice on how to proceed, these three gentlemen came through. I would also like to thank Dr. Fred D. Garber, Dr. Marian K. Kazimierczuk, and Dr. Raymond E. Siferd for serving as members of my MS thesis defense committee. I would also like to thank this college for thinking so much of me to offer an assistantship so I could pursue my masters degree here at Wright State University.

1 Introduction

In today's technology, the miniaturization of electronics is accelerating. Components that once took up significant amounts of space, now occupy a fraction of an integrated chip (*IC*). With the inclusion of a discrete power supply, the overall product is very large when compared to the functional electronics it powers. Such is the case with *MEMs.* What is needed is a dynamic power supply which co-exists along with the *MEMs* device. This power supply should be able to source or sink a high amount of current while adequately dealing with the high switching speed of *MEMs* devices.

The purpose of this thesis is to create an on-chip power supply that can handle 500 mA, operate as fast as possible, and if possible use one power rail. This concept represents a state-of-the-art technology that meets the following conditions. First, as designers push for more Design-For-Testability (*DFT*) on-chip, the use of an on-chip power supply will remove one more uncertainty from the design process. Second, this technique will help to merge different design paradigms together. When all the necessary fabrication technologies are available, customers will be able to custom design their *MEMs* devices along with the appropriate power supplies. This will bring the cost of these fabrications down to where a wide audience can purchase them.

2 Wideband Operational Amplifier

In this chapter the first phase of this thesis is undertaken. Section 2.1 shows a typical low-power open-loop gain ($A_{V_{OL}}$) for the Tanner[1] library operational amplifier (op-amp). Methodology issues in the design of this op-amp are seen in the remaining three sections. Section 2.2 provides development of the transconductance op-amp (non-buffered) along with a short example. In Section 2.3, a procedure for design is given along with the rationale behind it. Finally, Section 2.4 covers open-loop simulation which confirms the design developed in the previous two sections.

2.1 The Tanner Operational Amplifier

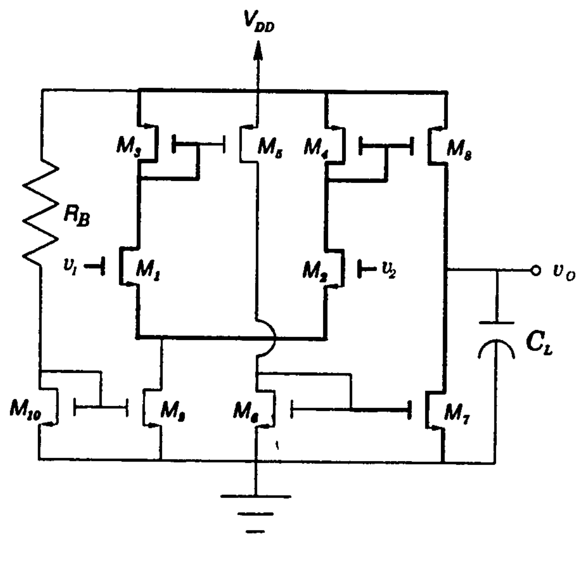


Figure 1: Tanner wideband op-amp.

While researching possible op-amps which might fulfill the purposes of this thesis, the Tanner wideband circuit shown in Figure 1 was chosen as the best. The main

Table 1: Transistor aspect ratio parameters for the Tanner circuit.

Transistor	Width (μm)	Length (μm)
M1	6	4
M2	6	4
M3	4	4
M4	4	4
M5	4	4
M6	4	4
M7	4	26
M8	4	26
M9	12	4
M10	12	4

reason behind this decision was the wide output voltage swing this circuit provides. This was especially crucial with the last requirement regarding only one power rail. This constraint, while not set in stone, hampered all the other one or two-stage op-amps under consideration. Seeing that only an op-amp with this kind of indifference to power rails, due to intelligent design, passed this constraint, a concerted effort was made to not violate it. The reference voltage, bias resistor, and load capacitor are all realizable within the CMOS process[4] and will not be covered. The dynamic

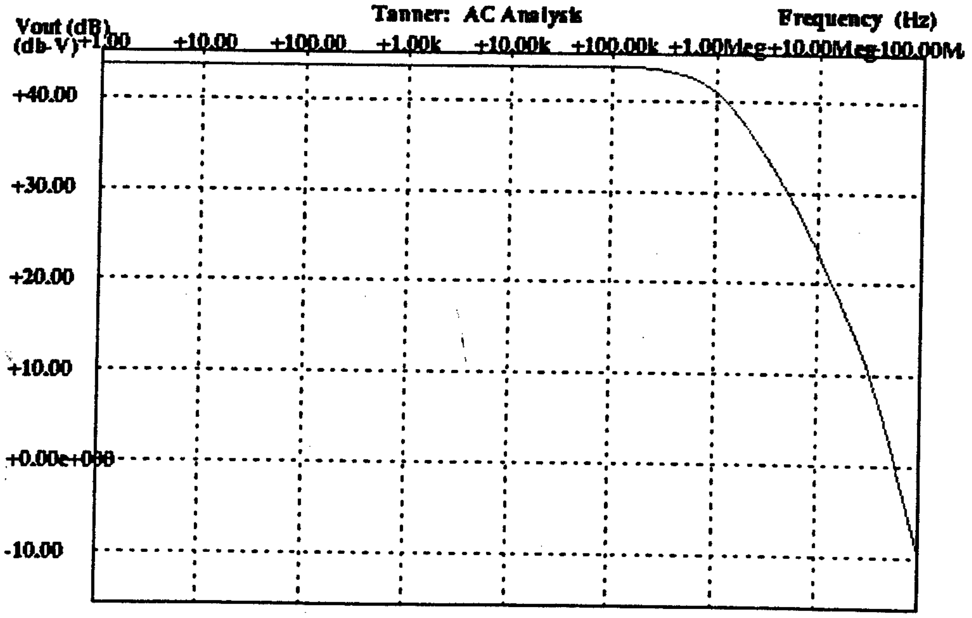


Figure 2: Voltage gain of op-amp in Figure 1.

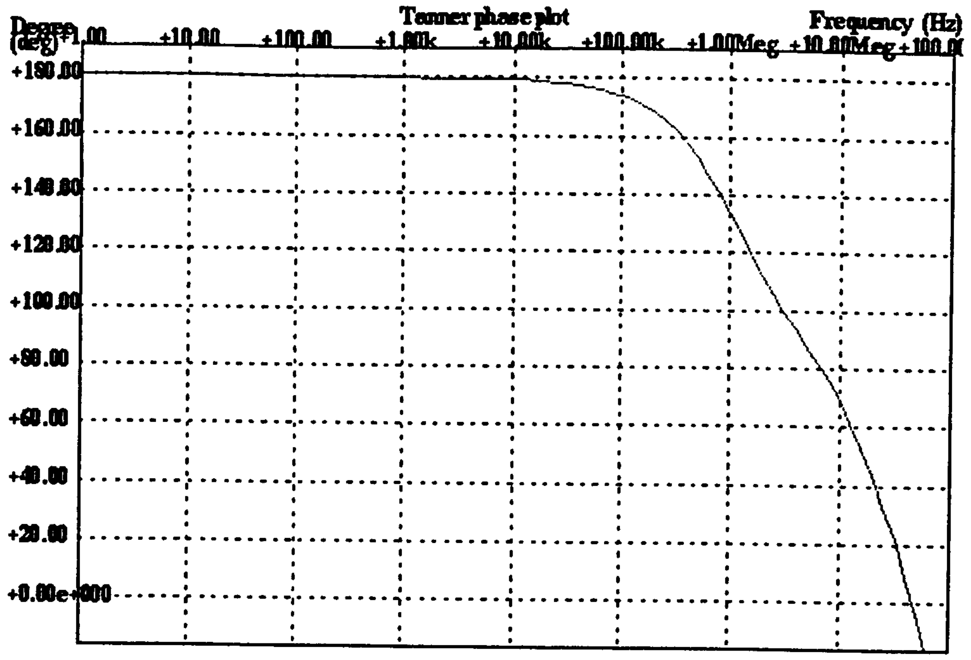


Figure 3: Phase plot of op-amp in Figure 1.

response generated by SPICE in Figures 2 and 3 was in response to parameters from an analog VLSI course[2] laid out in Table 1. It shows the typical low power response of the circuit. The gain of the op-amp is $\simeq$ 44 dB while the gain bandwidth product (GBP) is $\simeq$ 61 MHz while the phase predicts how the op-amp will act. From these figures, some speed boundaries show up. This circuit is designed around 2 μa. If more current is needed, the bandwidth of the circuit will shrink as it is traded off for more current handling capability.

2.2 Low-Frequency Small-Signal Analysis

Referring to Figure 1, although this is a one-stage amplifier, it is easier to explain its operation if we treat transistors M_1-M_4 as stage 1a and transistors M_7 and M_8 as stage 1b. Bias current for the circuit is controlled by the bias resistor R_B. The current handling capability of stage 1a is set up by the current mirror arrangement of M_9 and M_{10}. With proper design, the current provided will be split between the two legs of this stage. A network of current mirrors from stage 1a provides current to stage

1b. Current amplification between the two stages is dependent on the aspect ratios of the transitional transistors. To understand how this works, midband small-signal modeling must be done. Figure 4 shows a low-frequency small-signal model of stage

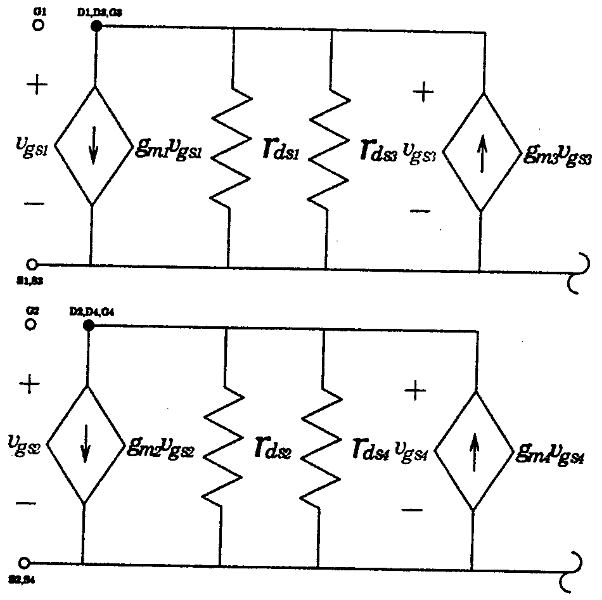


Figure 4: Stage 1a small-signal model.

1a. The dependent current source $g_{m3}v_{gs3}$ is driven by its own voltage v_{gs3}. Therefore, applying the source absorption theorem [3], the dependent current source may be replaced by a resistance $\frac{1}{g_{m3}}$. The same procedure can be applied to the dependent current source $g_{m4}v_{gs4}$. The resulting simplified small-signal model of stage 1a is depicted in Figure 5. Since

$$r_{ds1} \parallel r_{ds3} \gg \frac{1}{g_{m3}},$$

$$i_3 \simeq i_1 = g_{m1}v_{gs1}. \tag{1}$$

Likewise, since

$$r_{ds2} \parallel r_{ds4} \gg \frac{1}{g_{m4}},$$

$$i_4 \simeq i_2 = g_{m2}v_{gs2}. \tag{2}$$

Figure 6 shows a low-frequency small-signal model of stage 1b. It can be seen from

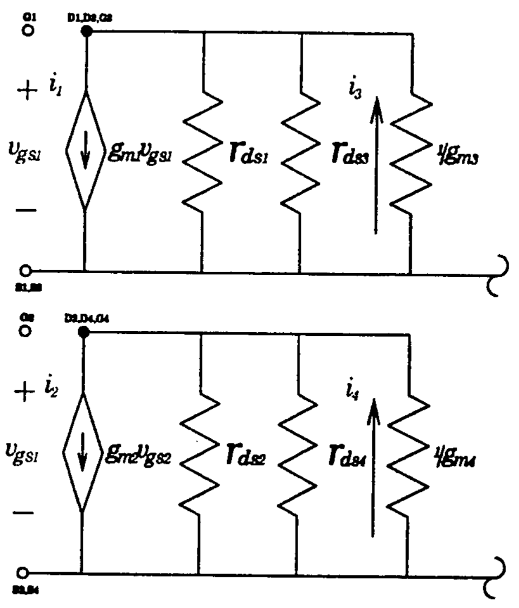


Figure 5: Stage 1a simplified small-signal model.

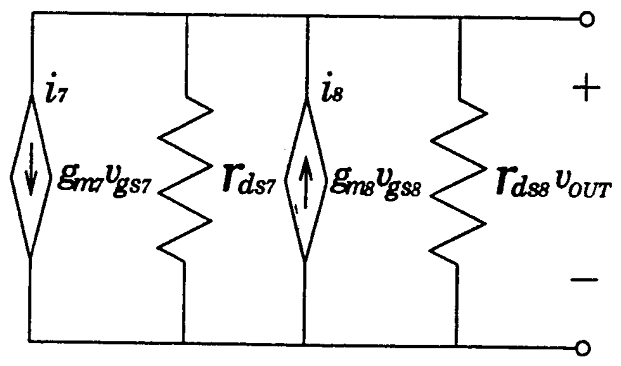


Figure 6: Stage 1b small-signal model.

this figure that

$$v_{OUT} = (i_8 - i_7)\,(r_{ds7} \parallel r_{ds8})\,. \tag{3}$$

Referring to Figure 1 and recalling Equation (1) and Equation (2),

$$i_8 = i_4 \left(\frac{\mathrm{W}_8}{\mathrm{W}_4}\right) \tag{4}$$

and

$$i_7 = i_6 \left(\frac{\mathrm{W}_7}{\mathrm{W}_6}\right). \tag{5}$$

Note that i_5 equals i_6 and if W_5 equals that of W_3, then

$$i_7 = i_3 \left(\frac{\mathrm{W}_7}{\mathrm{W}_6}\right). \tag{6}$$

Two assumptions made are

$$\frac{\mathrm{W}_8}{\mathrm{W}_4} = \frac{\mathrm{W}_7}{\mathrm{W}_6} \tag{7}$$

and

$$g_{m1} = g_{m2} = g_{md}. \tag{8}$$

Referring to Figure 1,

$$v_{ID} = v_1 - v_2. \tag{9}$$

Hence,

$$v_{OUT} = \left(g_{m2} v_2 \frac{\mathrm{W}_8}{\mathrm{W}_4} - g_{m1} v_1 \frac{\mathrm{W}_7}{\mathrm{W}_6}\right)(r_{ds7} \parallel r_{ds8})\,. \tag{10}$$

This reduces down to

$$v_{OUT} = -\left(v_{ID} g_{md} \frac{\mathrm{W}_7}{\mathrm{W}_6}\right)(r_{ds7} \parallel r_{ds8})\,. \tag{11}$$

Therefore, the low-frequency open-loop gain A_{vo} is

$$A_{vo} = \frac{v_{OUT}}{v_{ID}} = -g_{md} \frac{\mathrm{W}_7}{\mathrm{W}_6}(r_{ds7} \parallel r_{ds8}) \tag{12}$$

with

$$g_{md} = 2\sqrt{K I_{D1}} \tag{13}$$

and

$$r_{ds7} \parallel r_{ds8} = R_L = \frac{1}{(\lambda_n + \lambda_p)\, I_{D7}}. \tag{14}$$

2.2.1 Example

For the following given values: $I_{D1} = 200\ \mu\text{A}$, $I_{D7} = 5$ mA, $K_n = 4.4782 \times 10^{-5}\frac{\text{A}}{\text{V}^2}$, $K_p = 1.5844 \times 10^{-5}\frac{\text{A}}{\text{V}^2}$, $\lambda_n = 0.01\ \text{V}^{-1}$, $\lambda_p = 0.02\ \text{V}^{-1}$, and $\frac{\text{W}_7}{\text{W}_6} = 25$, the transconductance of M_1 and M_2 is

$$g_{md} = 2\sqrt{K_n I_{D1}} = 2\sqrt{4.4782 \times 10^{-5} \times 200 \times 10^{-6}} = 189.277\,\frac{\mu\text{A}}{\text{V}}.$$

The effective load resistance is

$$R_L = r_{ds7} \parallel r_{ds8} = \frac{1}{(\lambda_n + \lambda_p)\, I_{D7}} = \frac{1}{(0.01 + 0.02)\, 5 \times 10^{-3}} = 6.67\ \text{k}\Omega.$$

The theoretical low-frequency small-signal open-loop gain A_{vo} is

$$A_{vo} = -g_{md}\frac{\text{W}_7}{\text{W}_6}\left(R_L\right) = -189.277 \times 10^{-6} \times 25 \times 6.67 \times 10^3 = 31.562\,\frac{\text{V}}{\text{V}}$$

which gives $\mid A_{vo} \mid_{dB} = 30$ dB. In Section 2.4, the above circuit will be analyzed to see if simulation results agree with theoretical results.

2.3 Design Procedure

To design an op-amp, the major regions of concern are the power rails and the output power characteristics. In this case, one of the power rails is ground. This limits the output voltage swing potential by half with respect to normal dual non-zero power rail configurations. Another concern is the length of the transistor channels. Channel lengths throughout the design have to be big enough to ensure that the channel modulation (λ) is sufficiently small. This is different from digital designs where the length of λ is as small as possible for rise and fall time purposes. In the analog world, the designer is more worried about the slope in the saturation region. The slope is inversely related to channel length. If this is taken into account, calculated open-loop gain will be close to that provided by SPICE simulations. There are only two equations that must be used in designing this op-amp. For CMOS devices, the drain current (i_D) is expressed as

$$i_D = \frac{\beta_n}{2}\left(v_{GS} - V_t\right)^2\left(1 + \lambda_n v_{DS}\right) \tag{15}$$

for NMOS transistors and as

$$i_D = \frac{\beta_p}{2}\left(v_{SG} - \mid V_t \mid\right)^2\left(1 + \lambda_p v_{SD}\right) \tag{16}$$

for PMOS transistors. In determining the associated aspect ratios , Equations (15) and (16) become

$$\frac{W_n}{L_n} = \frac{2 \times i_D}{\left(v_{GS} - V_t\right)^2\left(1 + \lambda_n v_{DS}\right)} \tag{17}$$

and

$$\frac{W_p}{L_p} = \frac{2 \times i_D}{\left(v_{SG} - \mid V_t \mid\right)^2\left(1 + \lambda_p v_{SD}\right)}. \tag{18}$$

In a design environment, three of the five variables within these design equations are known. For example, in Equation (17), v_{GS} and v_{DS} are unknown but for the transistor to operate in the saturation region, these variables are restricted by

$$v_{DS} \geq v_{GS} - V_t. \tag{19}$$

Similarly, for a PMOS transistor, variables v_{SG} and v_{SD} are restricted by

$$v_{SD} \geq v_{SG} - \mid V_t \mid . \tag{20}$$

Starting with stage 1b, there is only one unknown since both v_{DS} and v_{SD} are known. The designer will then select the quiescent output voltage. Looking at M_7, the circuit designer will select v_{GS7} (actually just v_{G7} since source is ground) such that it will remain in saturation while still allowing minor voltage swings. The same will be done with M_8. Once these transistor width to length ratios are figured, the designer needs to choose a minimum length per transistor so the saturation region of operation is relatively flat. This length should be used throughout the design, providing a stable base upon which to use current mirror aspect ratios to design most of the other transistors (except for M_1 and M_2). Using Equations (4) and (5), the widths for M_6 and M_4 may be quickly calculated as long as Equation (7) holds true. By picking the same gate voltage for transistors M_3 and M_4, the widths of all the PMOS transistors not in stage 1b will be the same. When the design parameters are passed to the designer, the bias resistor (R_B) can be easily figured using Ohm's Law as long as V_{G9} is selected properly. With this voltage known, the widths of M_9 and M_{10} can be quickly found since there are no unknowns left in Equation (15). The only transistor widths to calculate are W_1 and W_2. These two do not have their sources connected to ground. This presents a problem that can be taken care of in the following manner: Let the source voltage of both of these transistors be labeled as v_x. Note that v_x has to be greater than or equal to v_{G9}. The problem is what to make v_{G1} and v_{G2}. It can be seen that

$$\begin{aligned} v_{DS1} &\geq v_{GS1} - V_t, \\ v_{G3} - v_x &\geq v_{G1} - v_x - V_t, \\ v_{G3} &\geq v_{G1} - V_t, \end{aligned}$$

and

$$v_{G1} \leq v_{G3} + V_t. \tag{21}$$

This will make the gate voltages higher than $v_{OUT_{Quiescent}}$ as long as the need for $\frac{W_n}{L_n}$ is to stay as small as possible. Therefore, for Example 2.2.1, with $v_{G7} = 1.5\,\text{V}$, $v_{G8} = 3.5\,\text{V}$, $v_{G1,2} = 3\text{V}$, and all L = 15 μm, the following results:

$$\begin{aligned} \text{W}_1 = \text{W}_2 &= 205\,\mu\text{m}, \\ \text{W}_3 = \text{W}_4 = \text{W}_5 &= 1356\,\mu\text{m}, \\ \text{W}_6 &= 270\,\mu\text{m}, \\ \text{W}_7 &= 6694\,\mu\text{m}, \\ \text{W}_8 &= 33255\,\mu\text{m}, \end{aligned}$$

$$W_9 = W_{10} \quad = \quad 6720\,\mu\text{m},$$

and

$$R_B = 6.67\,\text{k}\Omega.$$

Once a designer gets to this stage in the development of a circuit, certain analyses need to be done to ascertain if the circuit is working as theoretically derived.

2.4 Open-Loop Op-Amp Simulation

To provide empirical data to support the theoretical design of the wideband op-amp, the results from SPICE simulations are provided along with the reasons they are needed. The SPICE analyses of interest are DC, AC, and Transient.

2.4.1 DC Simulation

DC simulation is used to find the input voltage or current (voltage in this case) needed to result in the designed quiescent output voltage. In this case with V_{DD} and v_2 held constant at 5 V and 3 V respectively, v_1 is swept over a range of 0 V to 5 V. The goal of this simulation is to find what voltage v_1 needs to be to result in the quiescent output voltage equaling 2.5 V. As can be seen in Figure 7, the offset

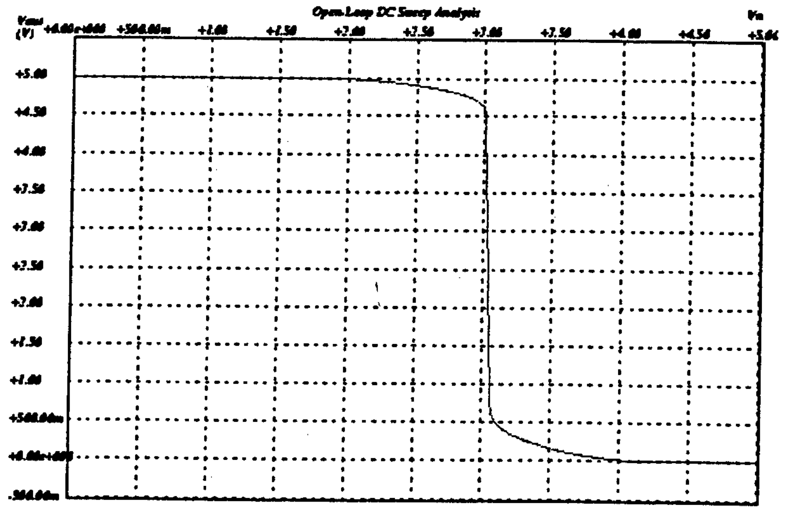


Figure 7: V_O vs. V_{in}.

voltage necessary to obtain the quiescent output voltage is 3.039 V.

2.4.2 AC Simulation

The second SPICE analysis performed is a frequency response or AC analysis. In this analysis, an AC signal is introduced into the circuit and is swept through a range of frequencies to see the range of usable operation. From Figure 8, it can be

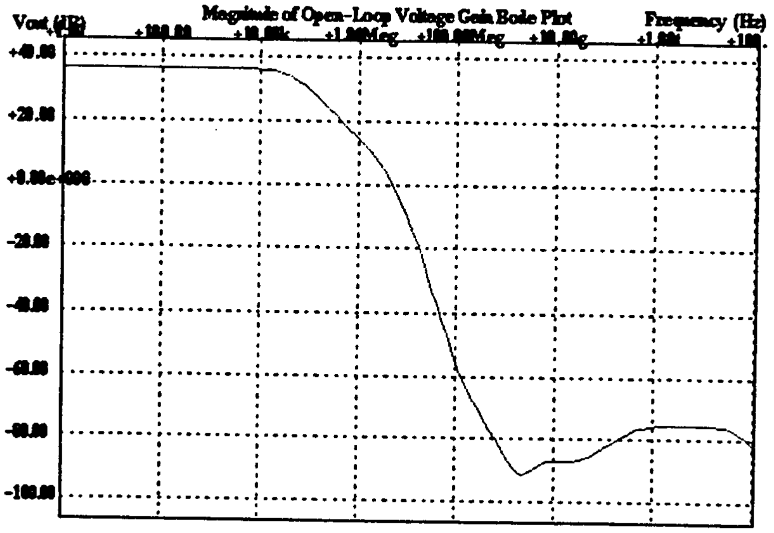


Figure 8: Magnitude of the open-loop voltage gain.

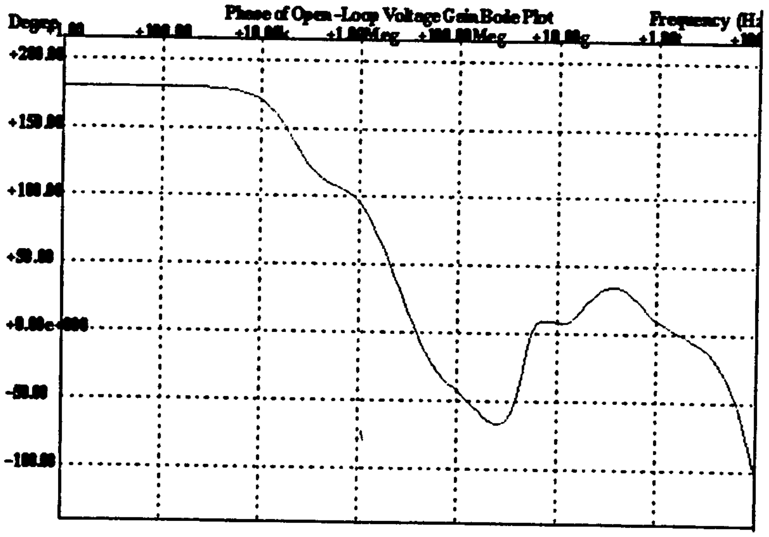


Figure 9: Phase of the open-loop voltage gain.

seen that this circuit's AV_{OL} is $\simeq$ 36 V/V. Also it can be seen from Figure 9, that this circuit, past $\simeq$ 5 MHz, starts to act as an inverting type circuit. The AV_{OL} closely corresponds to the value in Example 2.2.1. Around 650 kHz, the gain of the circuit starts to drop off or attenuates at a very rapid rate. This range of frequency is called the bandwidth (BW) of the circuit. This is not to be confused with the *GBP*. The *GBP* is the absolute maximum range of frequency that a circuit can operate properly in (ie., 5 MHz). The BW of a circuit can be increased to that of the *GBP* by decreasing the gain of the system through the use of negative feedback as discussed in Section 3.1.

2.4.3 Transient Simulation

The last simulation results come from various transient analyses. These analyses test the circuit's reaction in real time. There are five different situations which must be dealt with. These are step changes in V_{id}, V_{DD}, R_L, and i_O along with a response due to a δ input. V_{id} is defined as the difference between inputs V_p and V_n. In

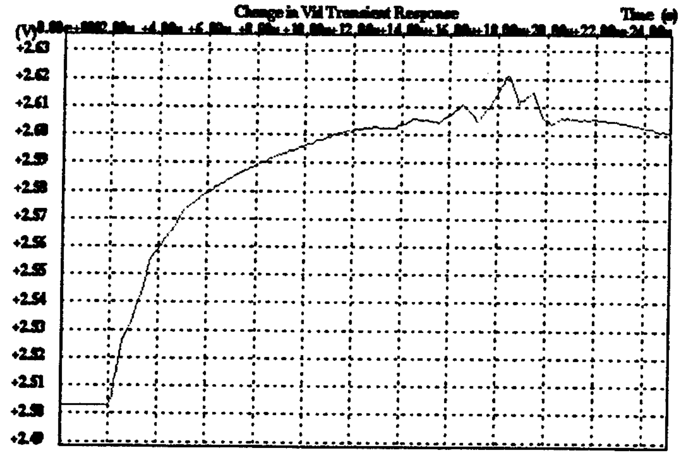


Figure 10: Transient open-loop V_O response due to a 2.5 μV change in V_{id}. This step change was initiated by deviating V_n while holding V_p steady.

Figure 10, the output generated is in response to an increase in the differential input voltage. In Figure 11, the output generated is in response to an increase in the power rail voltage. In Figure 12, the output generated is in response to an increase in the load resistance. In Figure 13, the output generated is in response to an increase in the load current. In Figure 14, it can be seen that the circuit is not very stable when a δ signal is introduced. In all these situations the addition of negative feedback

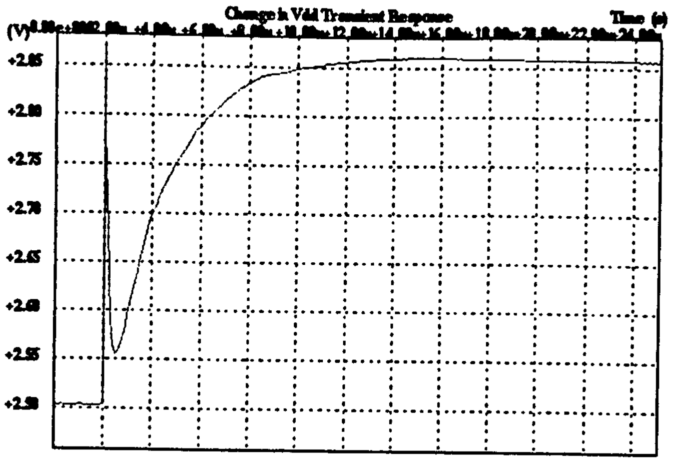


Figure 11: Transient open-loop V_O response due to a 1 V pulse in V_{DD}.

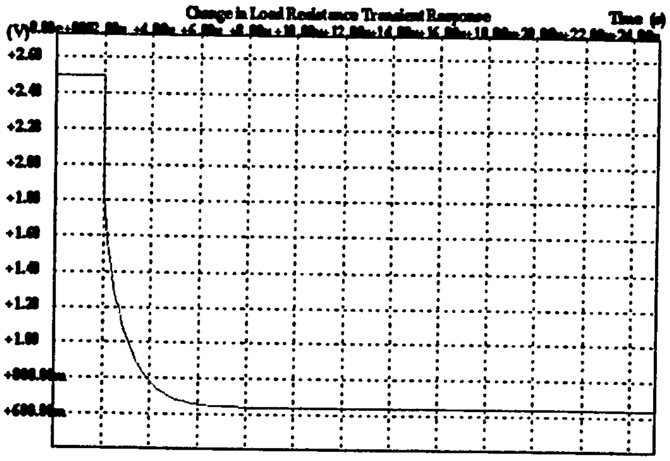


Figure 12: Transient open-loop V_O response due to a 1.1 kΩ introduction of R_L. This corresponds to a 2.5 mA load.

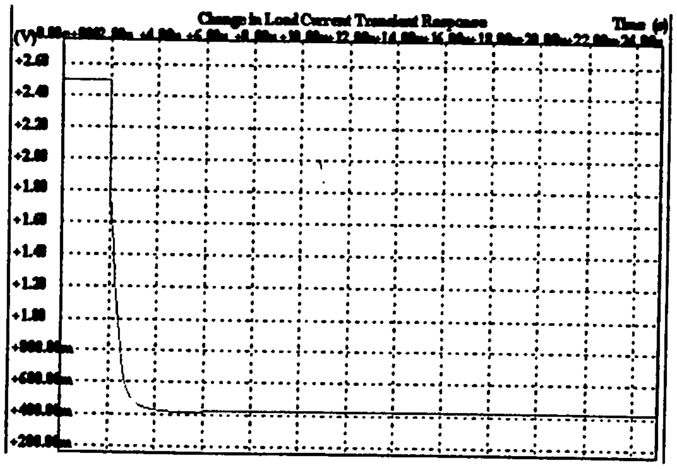


Figure 13: Transient open-loop V_O response due to a 2.5 mA increase in i_O.

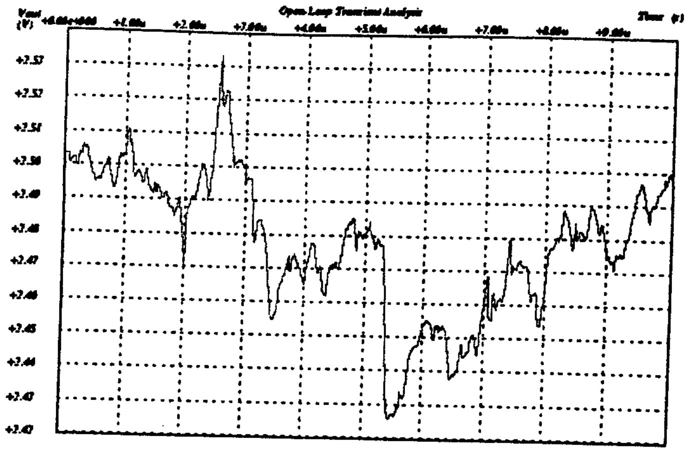


Figure 14: Transient open-loop V_O response due to 2.5 mV $\delta(t)$.

will greatly improve these response characteristics. This topic will be covered in Chapter 3.

3 Closed-Loop Results

Using the op-amp developed in Chapter 2, a beta network needs to be used to gain a stable output voltage. Recall that open-loop gain can be defined as

$$A_{V_{OL}} \equiv \frac{v_O}{v_S}. \tag{22}$$

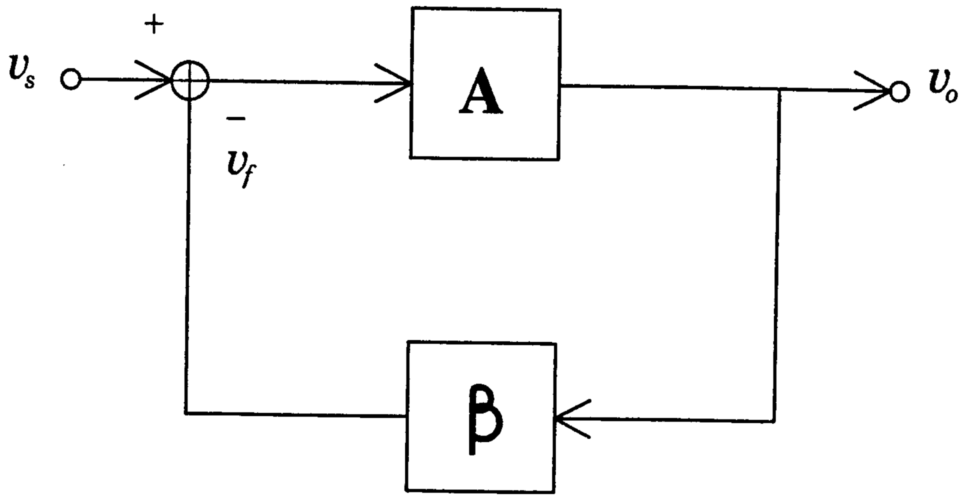


Figure 15: Block diagram of single closed-loop system.

The closed-loop gain depicted in Figure 15 is the basic negative feedback network where

$$A_{V_{CL}} \equiv \frac{A_{V_{OL}}}{1 + A_{V_{OL}}\beta} = \frac{1}{\frac{1}{A_{V_{OL}}} + \beta}. \tag{23}$$

As long as $A_{V_{OL}}$ is large, $A_{V_{CL}} \simeq \frac{1}{\beta}$. Therefore, it is important to select the proper beta network to control the open-loop circuit. After developing various negative feedback networks to control the open-loop circuit in Section 3.1, one best suited to the objectives stated earlier will be selected. In Section 3.2 the process of closed-loop testing using SPICE is introduced. These DC analysis simulations provide data to support the best source voltage - beta feedback network pair to be used.

3.1 Configuration of Various Negative Feedback Circuits

The configuration seen in Figure 16 is called a buffer or unity-gain circuit. Using

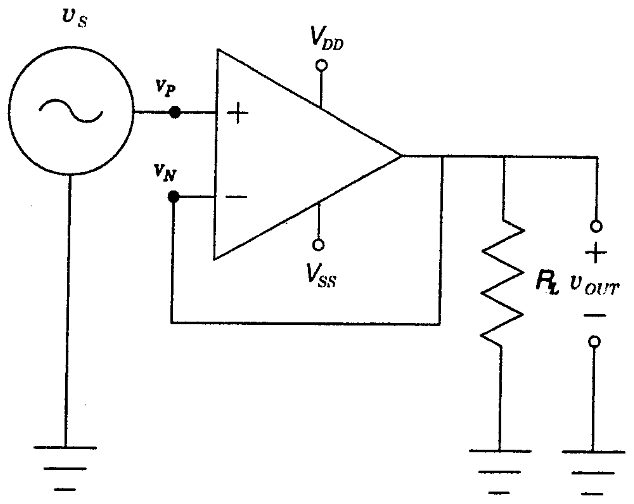


Figure 16: Buffer circuit.

nodal analysis, it is assumed that $v_P = v_N$ and current into the op-amp is zero. From this, it is known that

$$v_S = v_P = v_N = v_{OUT}.$$

Therefore,

$$A_{V_{CL}} \equiv \frac{v_{OUT}}{v_S} = 1. \tag{24}$$

Another way to look at this is $A_{V_{CL}} = \frac{1}{\beta}$ with $\beta = 1$. An advantage of this circuit is that the BW does not change from open-loop to closed-loop. Also, this circuit can be used to isolate loading effects from the source voltage. The disadvantage is that there is no amplification provided.

The negative feedback configuration set up by the resistors R_F and R_I in Figure 17 is called an inverter. To give validity to this name, nodal analysis will be used. Since current cannot flow into the op-amp, it is known that the current that flows through

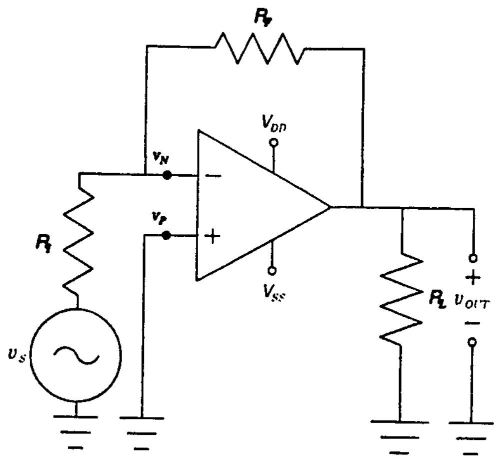


Figure 17: Inverter circuit.

R_I will also flow through R_F. Let node $v_N = V_A$. Using this information,

$$\frac{V_A - v_S}{R_I} + \frac{V_A - v_{OUT}}{R_F} = 0. \tag{25}$$

Since $V_A = 0$

$$\begin{aligned} \frac{-v_{OUT}}{R_F} &= \frac{v_S}{R_I}, \\ v_{OUT} &= -v_S \frac{R_F}{R_I}, \end{aligned}$$

and

$$A_V \equiv \frac{v_{OUT}}{v_S} = -\frac{R_F}{R_I}. \tag{26}$$

The advantage of this circuit is that voltage inversion can be done. The disadvantage of this circuit is that two power rails are required instead of one.

Another configuration used is called the non-inverting amplifier. This is depicted in Figure 18. Let $v_N = V_A$. Using V_A as the node of question, nodal analysis will result in

$$\frac{V_A}{R_I} + \frac{V_A - v_{OUT}}{R_F} = 0. \tag{27}$$

Rearranging gives

$$\begin{aligned} \frac{v_{OUT}}{R_F} &= \frac{V_A}{R_I} + \frac{V_A}{R_F}, \\ \frac{v_{OUT}}{R_F} &= V_A \frac{R_F + R_I}{R_F R_I} \end{aligned}$$

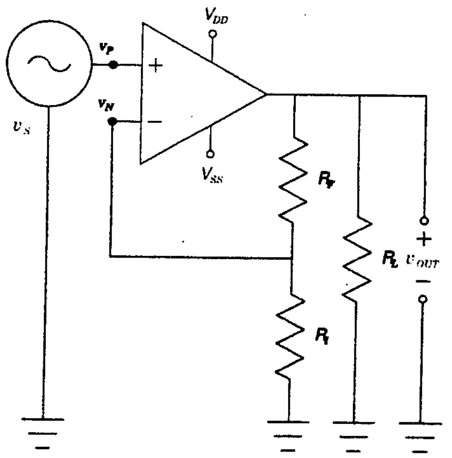


Figure 18: Non-inverting amplifier.

and

$$v_{OUT} = V_A \frac{R_F + R_I}{R_I}. \tag{28}$$

Knowing that $V_A = v_S$,

$$A_V \equiv \frac{v_{OUT}}{v_S} = \frac{R_F + R_I}{R_I} = 1 + \frac{R_F}{R_I}. \tag{29}$$

The advantage of this circuit arrangement is that only one power rail is needed if the input signal remains non-negative. As with the inverting amplifier, as BW increases, A_V decreases.

The last configuration that will be examined is that of a differential amplifier arrangement. Figure 19 shows a differential amplifier configuration. To fully analyze it, superposition along with nodal analysis will be used. In the first case, v_1 will be set to zero. As can be seen in Figure 20, using a simple voltage divider,

$$v_{2'} = v_2 \frac{R_F}{R_F + R_I}. \tag{30}$$

Also by nodal analysis,

$$\frac{V_A}{R_I} + \frac{V_A - v_{OUT'}}{R_F} = 0. \tag{31}$$

Rearranging gives

$$\frac{v_{OUT'}}{R_F} = \frac{V_A}{R_I} + \frac{V_A}{R_F}.$$

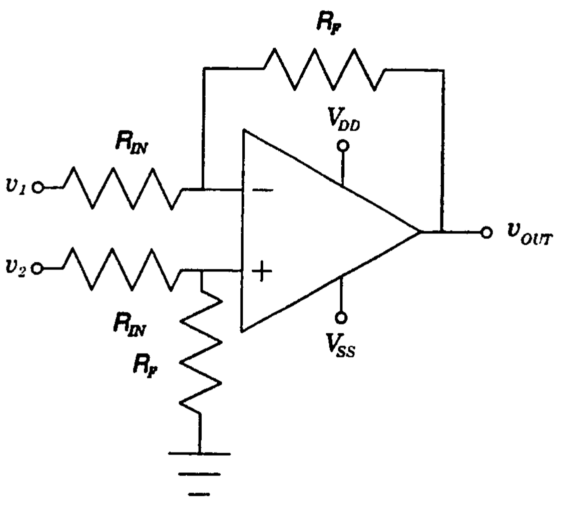


Figure 19: Differential amplifier.

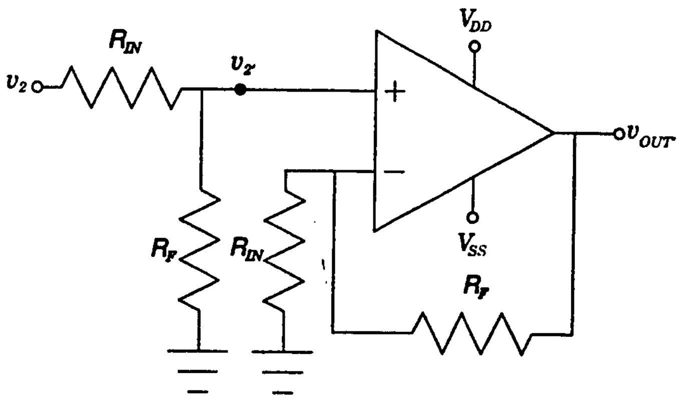


Figure 20: Superposition with v_1 set to zero.

Solving for $v_{OUT'}$ gives

$$v_{OUT'} = V_A \left(\frac{R_F + R_I}{R_I} \right). \tag{32}$$

Seeing that V_A equals $v_{2'}$, by substituting Equation (30) into Equation (32),

$$v_{OUT'} = v_2 \left(\frac{R_F + R_I}{R_I} \right) \left(\frac{R_F}{R_F + R_I} \right) = v_2 \frac{R_F}{R_I}. \tag{33}$$

The second case for superposition is when $v_2 = 0$. Figure 21 is the basic inverting

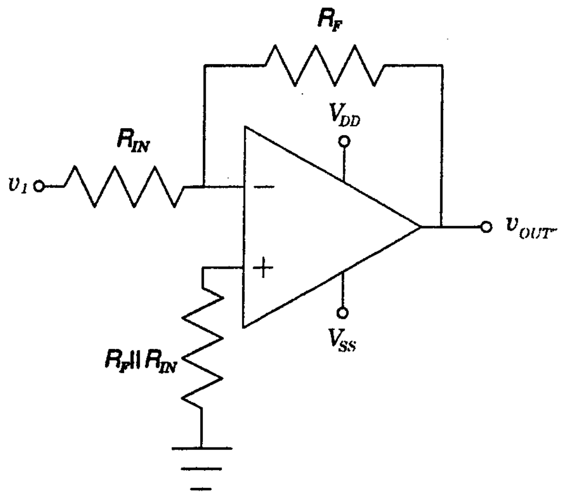


Figure 21: Superposition with $v_2 = 0$.

amplifier configuration analyzed earlier. Thus, it is known that

$$v_{OUT''} = -v_1 \frac{R_F}{R_I}. \tag{34}$$

Using Equations (30) and (34),

$$v_{OUT} = v_{OUT'} + v_{OUT''} \tag{35}$$

and

$$= v_2 \frac{R_F}{R_I} - v_1 \frac{R_F}{R_I}$$

and

$$= (v_2 - v_1) \frac{R_F}{R_I}. \tag{36}$$

Replacing $v_S = v_2 - v_1$ gives

$$v_{OUT} = v_S \frac{R_F}{R_I}. \tag{37}$$

The advantage of this amplifier configuration is that the gain is very high while the trade-off is a small usable BW.

From the above analyses, combined with the objectives of this thesis, only one amplifier circuit can be used, the non-inverting amplifier configuration. Now that a closed-loop circuit is ready, simulation analysis needs to be performed to see how the circuit will react under realistic real-time problems. This analysis will be covered in Section 3.2.

3.2 Non-Inverting Op-Amp DC Analysis

This section covers the basic DC analyzes necessary to assure that the closed-loop circuit will operate under the given design parameters. In order to properly design the closed-loop system developed thus far, there are a couple of variables that need to be analyzed. For the non-inverting amplifier arrangement, the controlling equation is

$$v_{OUT}^{\leftrightarrow} = v_S\left(1 + \frac{R_F}{R_I}\right). \tag{38}$$

It can be seen that since v_{OUT} is supposed to be constant, only v_S and the ratio of R_F and R_I can be manipulated. Referring to Figure 18, the only resistance that can be manipulated is R_I since R_F is isolated from ground. Therefore the equivalent input resistance is going to be in the form of parallel resistances or just R_I worst case.

The purpose of this closed-loop analyzes is to select the most constant beta network for the given source voltage. For this, three different test circuits, with beta equal to 2.5, 2, and 1.2, will be tested with the goal of achieving an output voltage of 3 V. The corresponding source voltages are 1.2 V, 1.5 V, and 2.5 V.

For no-load conditions, all three of the beta test circuits came out as expected. Figure 22 shows the response curve that provides evidence that $A_{V_{CL}} = 2$ V/V.

Using a circuit discussed later in Chapter 4, loading parameters were found for both full and half loading conditions. For half-load conditions, all three beta test circuits' transfer functions had degraded performance. With beta equal to 1.2, the

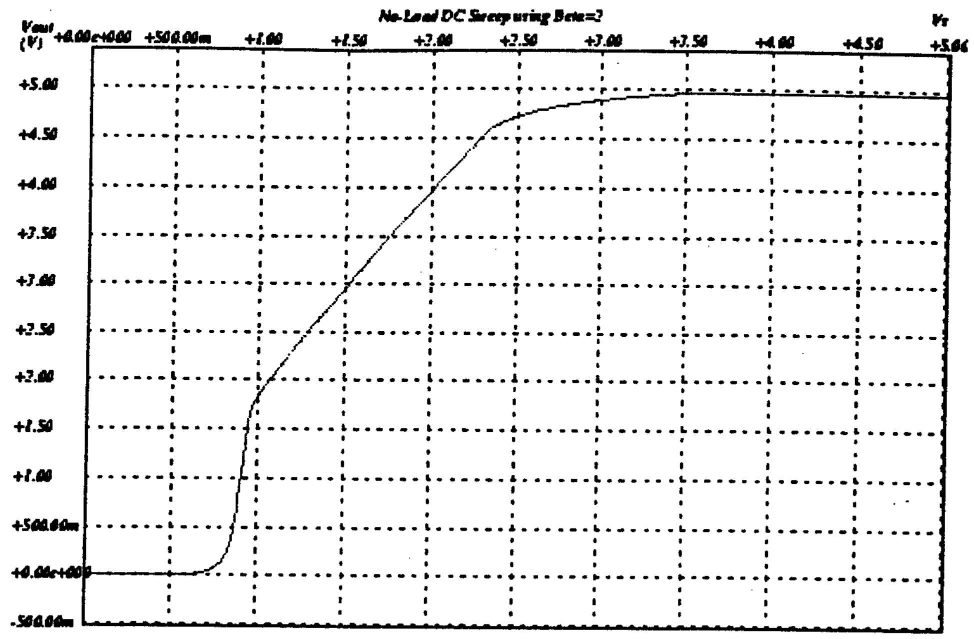


Figure 22: Closed-loop DC sweep with $A_{V_{CL}} = 2$ V/V from circuit in Figure 1.

closed loop gain of the circuit is seen in Figure 23 to be

$$A_{V_{CL}} \simeq 1.1 \frac{\text{V}}{\text{V}}.$$

Similarly, it can be seen that for the $\beta = 2$ test circuit, Figure 24 shows

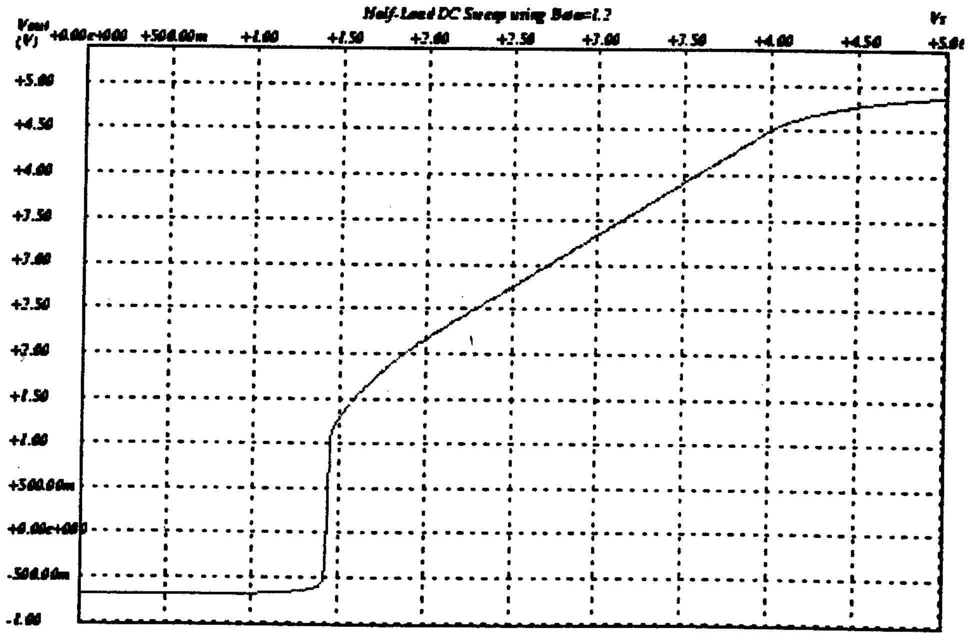


Figure 23: Half-load simulation with $A_{V_{CL}} = 1.2$ V/V from circuit in Figure 1.

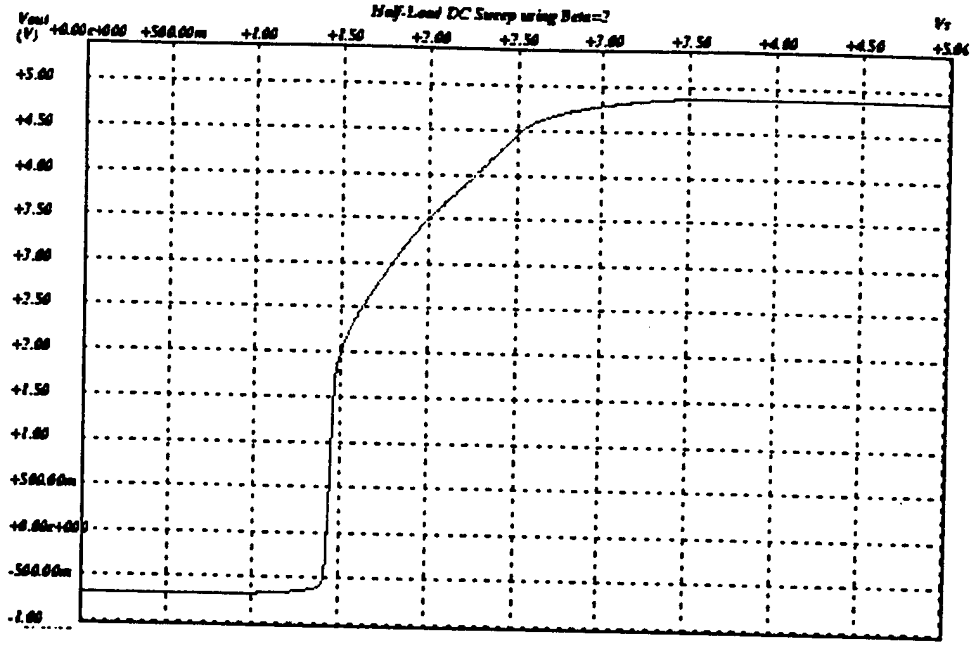


Figure 24: Half-load simulation with $A_{V_{CL}} = 2$ V/V from circuit in Figure 1.

$$A_{V_{CL}} \simeq 1.7 \frac{\mathrm{V}}{\mathrm{V}}.$$

The last of the three beta test circuits features $\beta = 2.5$. The results are featured in

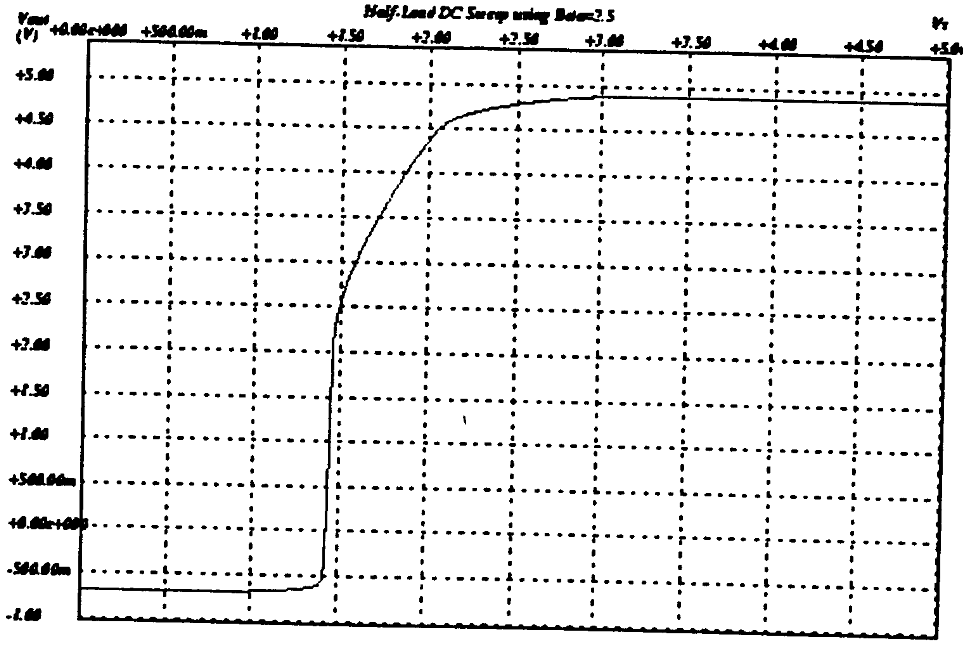


Figure 25: Half-load simulation with $A_{V_{CL}} = 2.5$ V/V from circuit in Figure 1.

Figure 25. This simulation result is

$$A_{V_{CL}} \simeq 1.9\,\frac{\text{V}}{\text{V}}.$$

Looking at these results, it can be predicted that the beta test circuit with $\beta = 1.2$ will have the best performance. With this network in place, the overall transfer function is still close to that of the theoretical results.

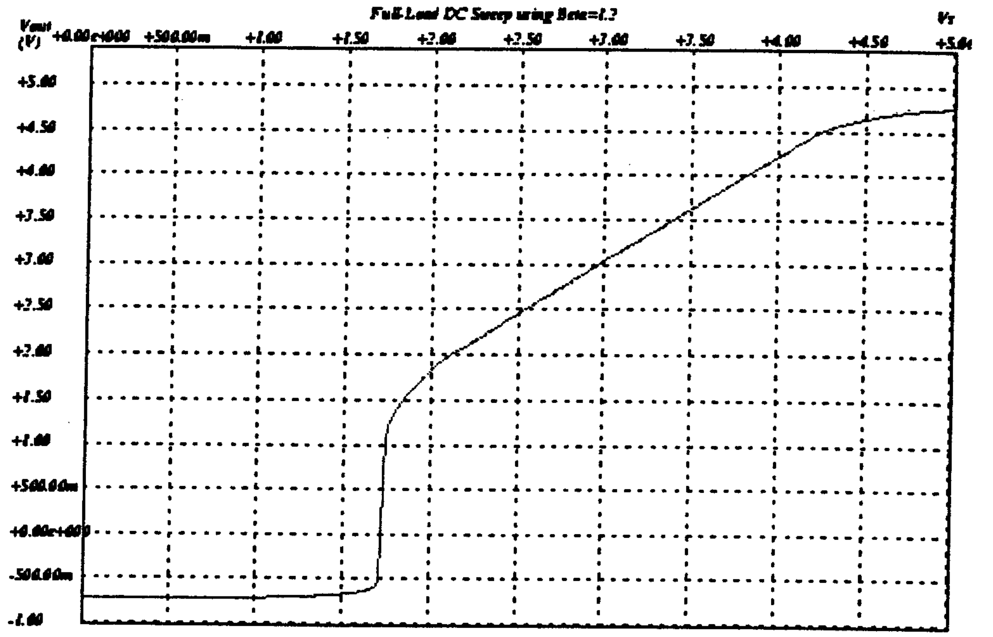


Figure 26: Full-load simulation with $A_{V_{CL}} = 1.2$ V/V from circuit in Figure 1.

As can be seen in Figures 26, 27, and 28, the transfer functions under full-load conditions are

$$\begin{aligned} A_{V_{CL_{\beta=1.2}}} &= 1.01\,\frac{\text{V}}{\text{V}}, \\ A_{V_{CL_{\beta=2}}} &= 1.5\,\frac{\text{V}}{\text{V}} \end{aligned}$$

and

$$A_{V_{CL_{\beta=2.5}}} = 1.63\,\frac{\text{V}}{\text{V}}.$$

These results provide more empirical data to support that the beta test circuit with $\beta = 1.2$ is the best choice of the three.

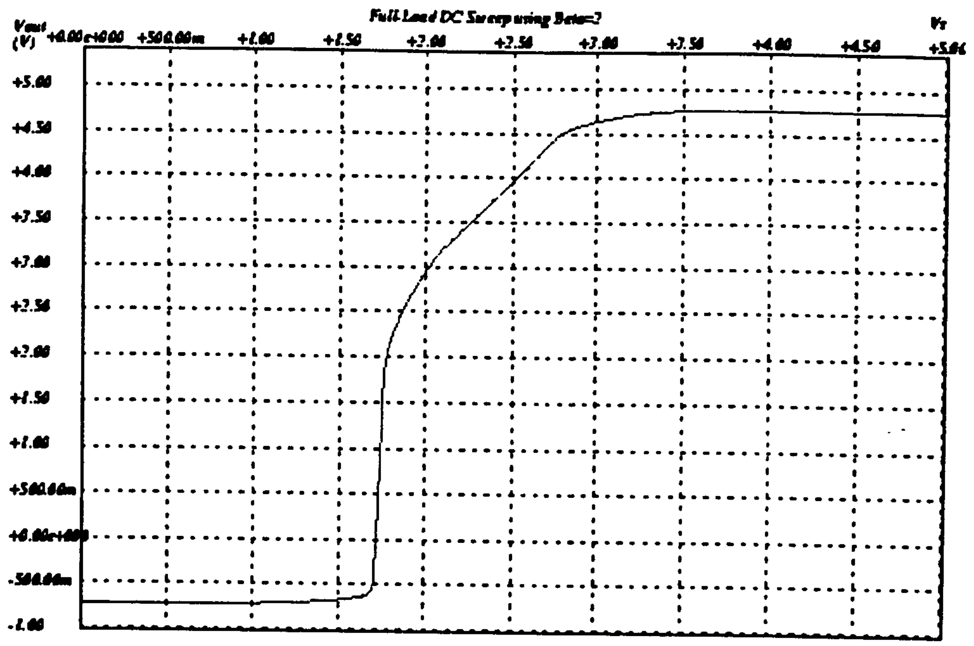


Figure 27: Full-load simulation with $A_{V_{CL}} = 2$ V/V from circuit in Figure 1.

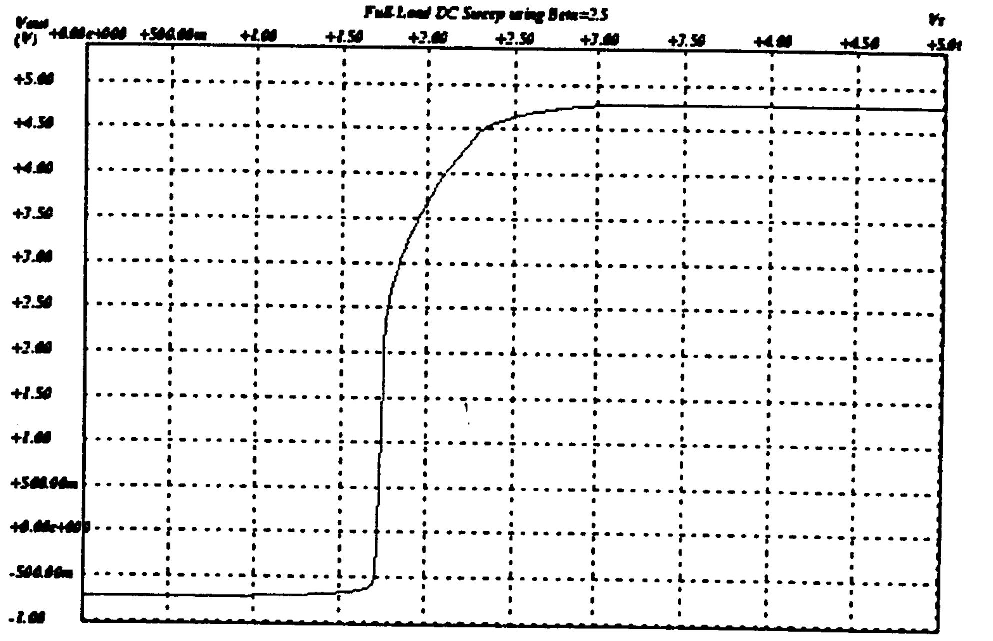


Figure 28: Full-load simulation with $A_{V_{CL}} = 2.5$ V/V from circuit in Figure 1.

Due to the nature of transconductance op-amps, the output stage has a output resistance of 6.67 kΩ. Ideally, the output resistance should be very close to zero for ideal results. In the case of multiple loads in parallel, there is a need for a much smaller output resistance and much greater current handling capability to work properly. To create a more "ideal" circuit, not to mention greater gain or drive capability, the designer has but two choices. The first is to use an unity-gain CMOS output stage. The second and better choice, based upon decisions made apparent next, is the use of a CMOS implementable BJT[4] output stage. Since one of the objectives of the thesis was to produce up to 500 mA, the choice between the above two choices becomes easy. A side-benefit of using an output stage consisting of a BJT is that the quiescent output voltage is lowered by approximately 0.7 V. Therefore the power supply will operate at 2.3 V instead of 3 V. Another reason not to use the CMOS output stage is that this stage would take up a significant portion of any chip die. This will be analyzed in Chapter 4.

4 Transient Testing

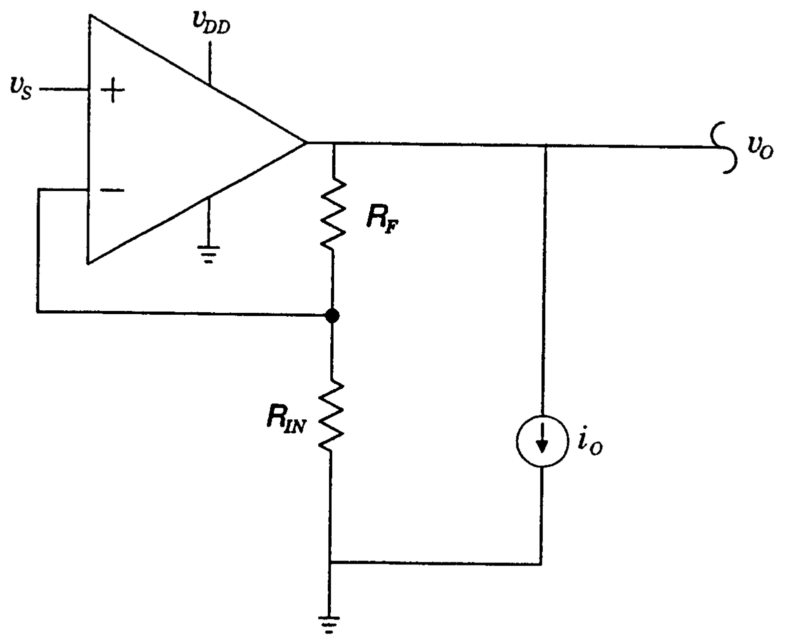


Figure 29: Load finder circuit.

When testing a circuit under different loading conditions, a way has to be created to find out the exact loading parameters so accurate testing can commence. Normally what is needed are the loading conditions that correspond to half-load and full-load. Figure 29 shows a circuit that will accurately determine not only the output resistance at a particular loading condition, but it will also provide the corresponding output voltage (this is due to the nature of transconductance op-amps). Once these parameters or metrics are found, transient testing can proceed. Clearly, with the addition of a current source acting as the load for the circuit, the designer can easily find out the output voltage and therefore by Ohm's law, the output resistance.

Both 5 mA and 500 mA design circuits will be tested and compared to find out

Table 2: Half and full resistive loading for both the 5 mA and 500 mA ciruits.

Circuit	5 mA	500 mA
Resistive half load	1.1 $k\Omega$	6.8 Ω
Resistive full load	490 Ω	2.8 Ω

which circuit operates more efficiently. To get an output of 500 mA, an output stage

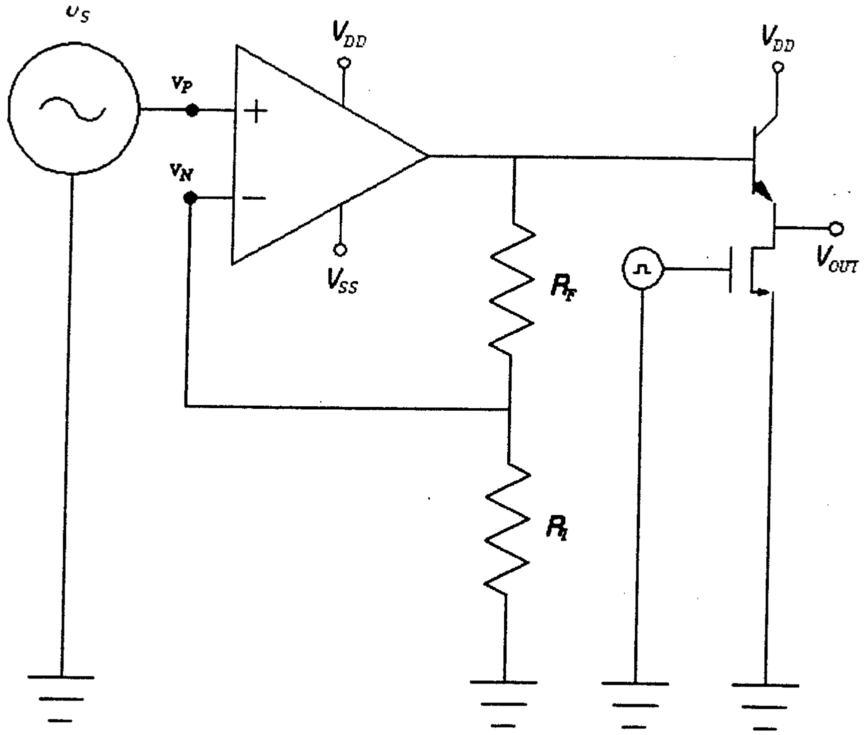


Figure 30: 500 mA circuit using a bjt output stage.

is introduced as seen in Figure 30. This output stage consists of a BJT implemented in CMOS and a biasing switch which only sinks 1 μA. For this thesis, the actual implementation of the bjt output stage is simulated by SPICE by assuming a value of β. The value of β used is 100. For all sections dealing with change in loading, Table 2 provides the necessary information.

In Section 4.1, the transient response is looked at to see how the circuit deals with momentary output surges while operating in steady-state. After that Section 4.2 describes how these circuits are tested under start-up conditions. The last section, Section 4.3 deals with the phenomenon of jumpy loads.

4.1 Perturbed Steady-State Response

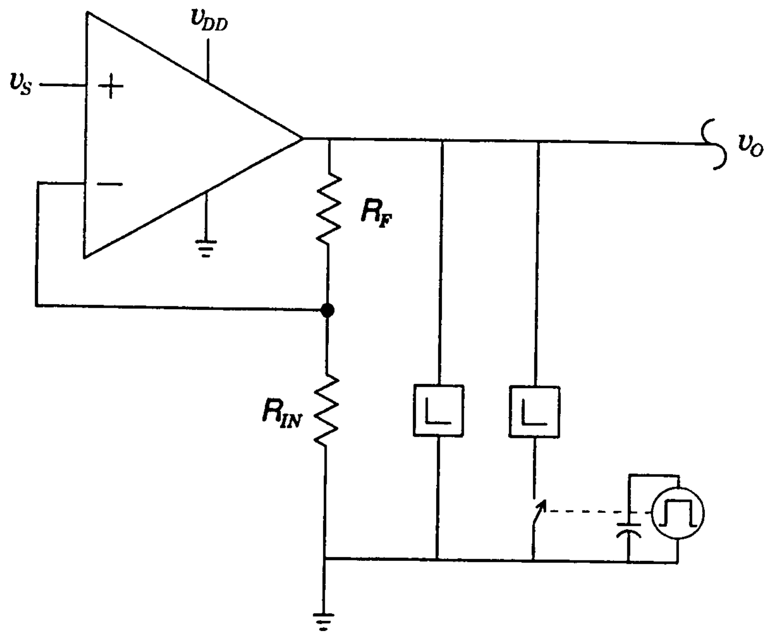


Figure 31: Test circuit to perform all transient analysis.

To test this type of analysis in SPICE, what is needed is to have the output momentarily pulse away from nominal value. The first condition to be tested is that of no-load steady-state operation with the load changing due to a momentary resistive load change. The method to test this kind of condition is pictorially explained in Figure 31.

As can be seen, Figure 32, Figure 33, Figure 34, and Figure 35 clearly point out that there is a difference between the two different ways to test for the same loading conditions. This can be seen in the difference of the voltage spikes between the two different testing ways. The next condition met is that of half-load steady-state.

Again, Figure 36, Figure 37, Figure 38, and Figure 39 support the observation that the response due to resistive versus current load changes create different responses. Therefore, the way testing is implemented has decisive bearing. The last condition in steady-state operation is that of full-load.

There are no new insights that this loading condition uncovers in Figures 40 to 43.

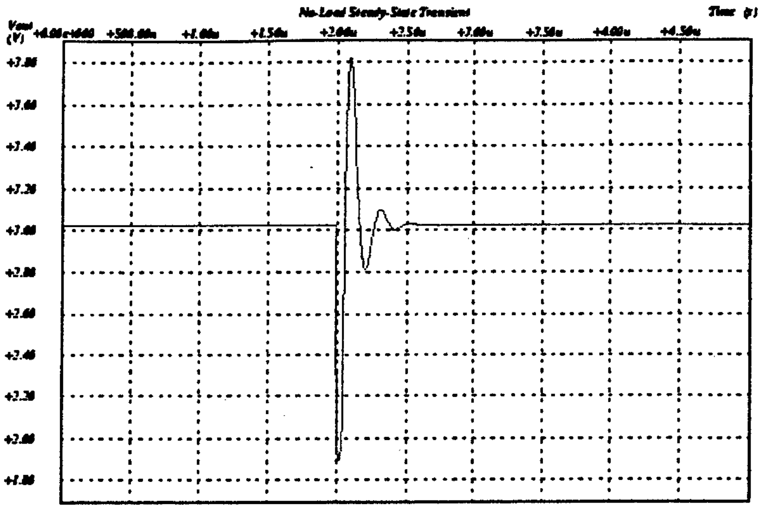


Figure 32: 5 mA circuit response to a momentary resistive load change from no-load to half-load and back as explained in Table 2.

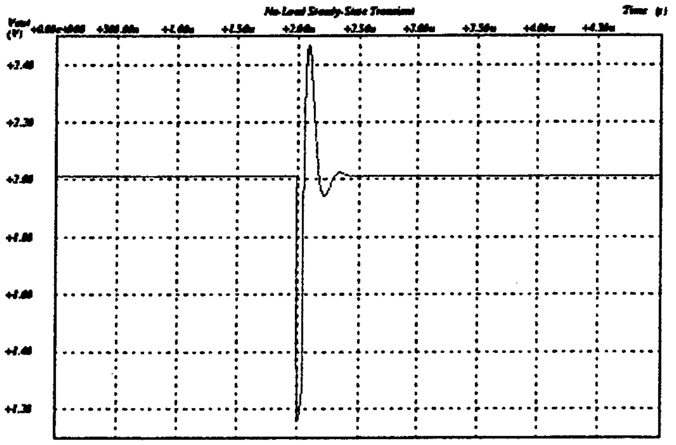


Figure 33: 500 mA circuit response to a momentary resistive load change from no-load to half-load and back as explained in Table 2.

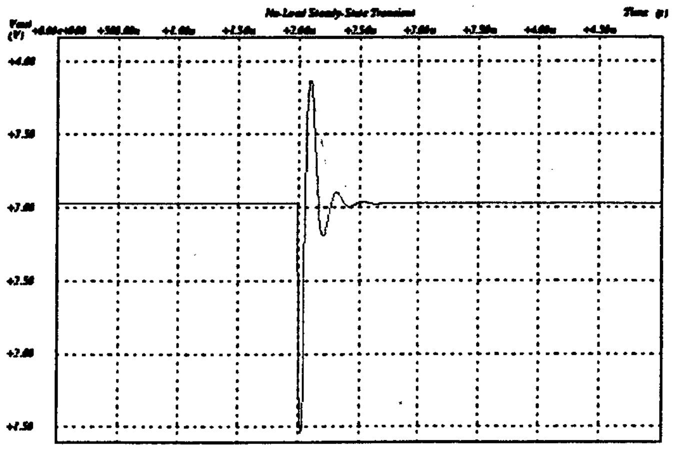


Figure 34: 5 mA circuit response to a momentary half-load current load introduction.

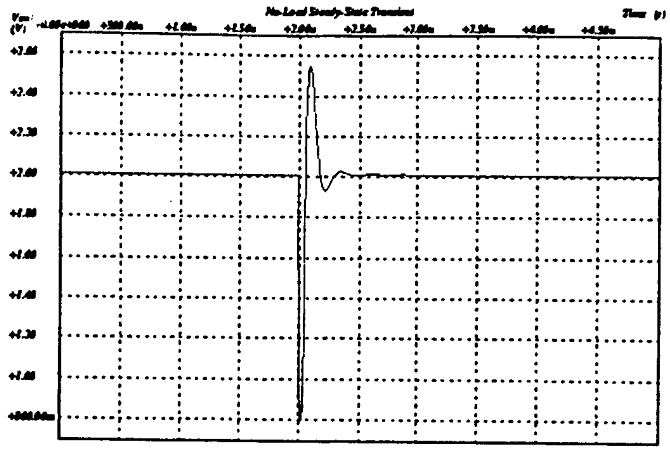


Figure 35: 500 mA circuit response to a momentary half-load current load introduction.

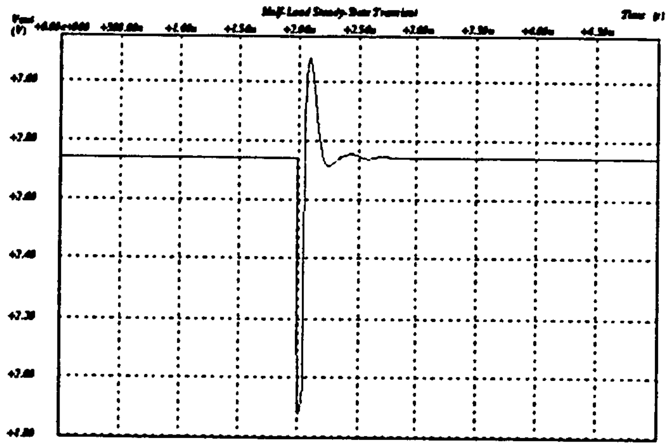


Figure 36: 5 mA circuit response to a momentary resistive load change from half-load to full-load and back as explained in Table 2.

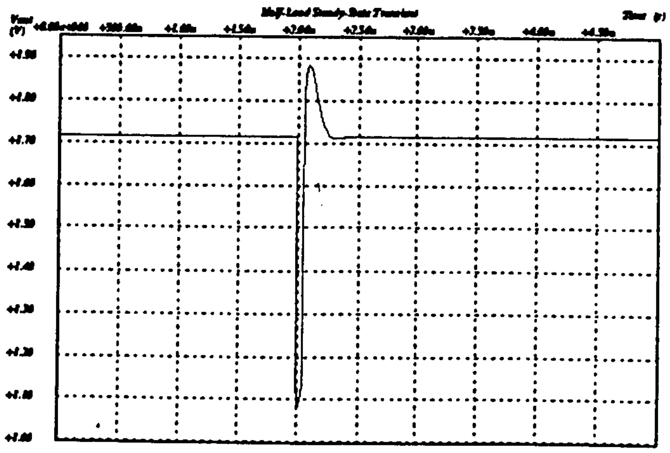


Figure 37: 500 mA circuit response to a momentary resistive load change from half-load to full-load and back as explained in Table 2.

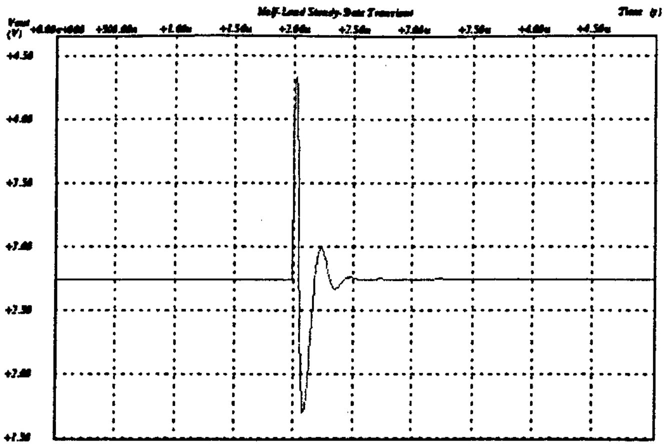


Figure 38: 5 mA circuit response to a momentary full-load current load introduction.

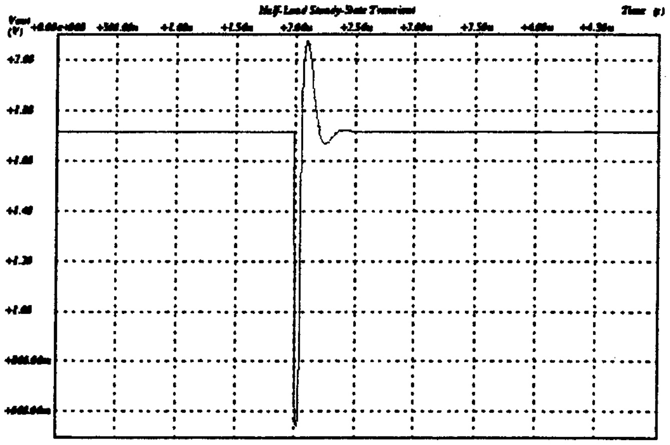


Figure 39: 500 mA circuit response to a momentary full-load current load introduction.

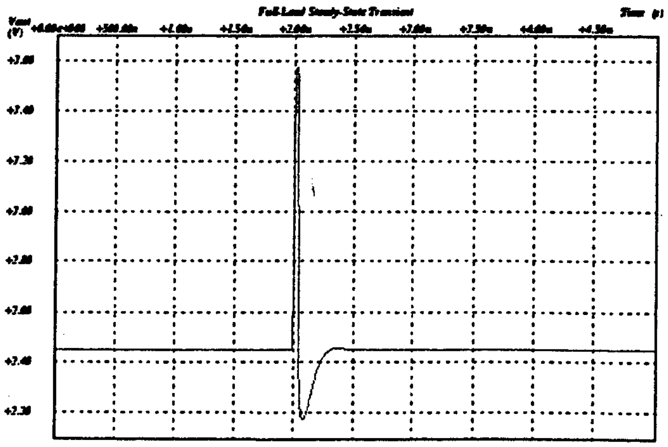


Figure 40: 5 mA circuit response to a momentary resistive load change from full-load to half-load and back as explained in Table 2.

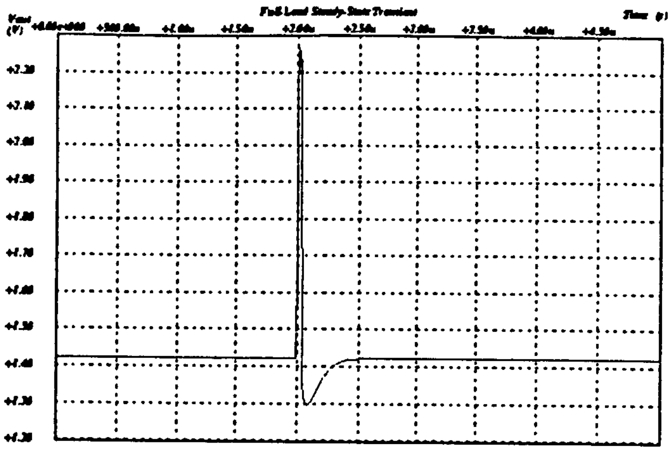


Figure 41: 500 mA circuit response to a momentary resistive load change from full-load to half-load and back as explained in Table 2.

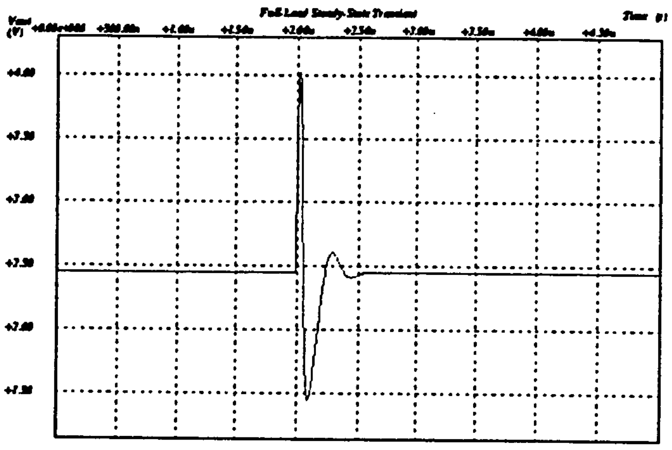


Figure 42: 5 mA circuit response to a momentary half-load current load reduction.

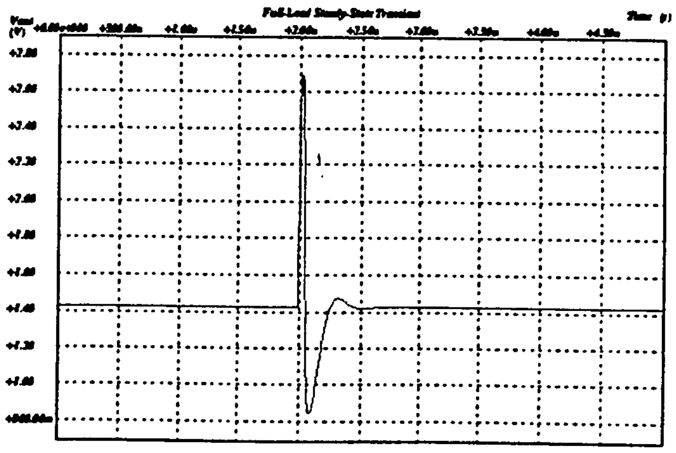


Figure 43: 500 mA circuit response to a momentary half-load current load reduction.

Only one thing can be observed on a global scale. While the response shapes are nearly identical between the two circuits under test, the 500 mA circuit does not have any negative voltage swing which might result in unexpected responses if that were to happen.

4.2 Start-Up Transient Testing

This section deals strictly with situations where the chip power supply and thus the source voltage loses power or is sleeping due to other operations. The first case is well understood and there are ways to try to deal with them such as a battery backup. The second case where the source voltage might be required to shutdown for a period of time is when dealing with mixed-signal applications. For this reason, the time it takes for the output voltage to stabilize will be the most delay usually seen by the chip. It can be seen in Figures 44 - 49 that as loading increases, the circuits

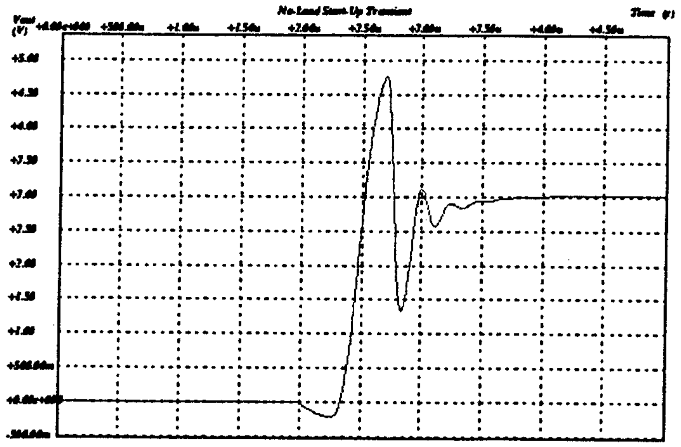


Figure 44: 5 mA circuit no-load source voltage start-up response.

become more over-damped.

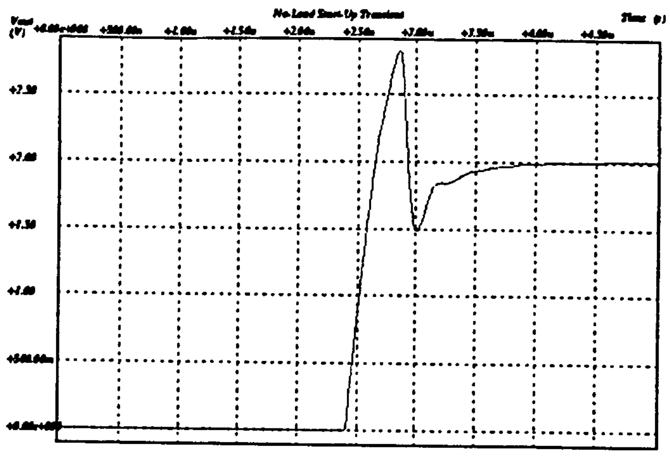


Figure 45: 500 mA circuit no-load source voltage start-up response.

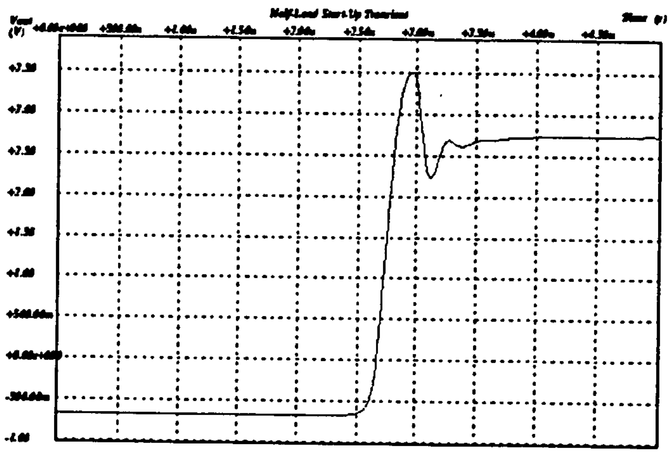


Figure 46: 5 mA circuit half-load source voltage start-up response.

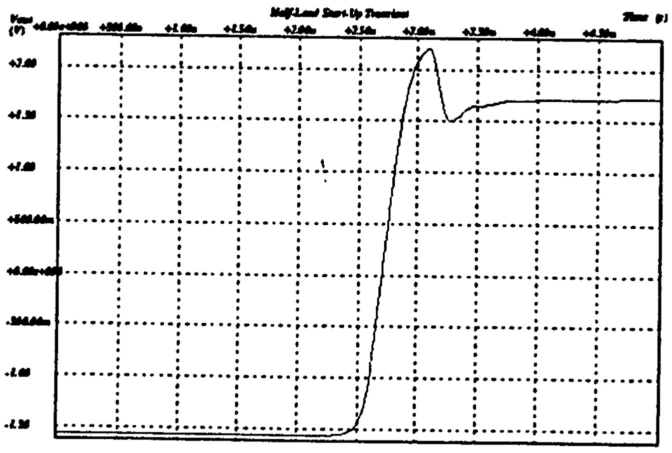


Figure 47: 500 mA circuit half-load source voltage start-up response.

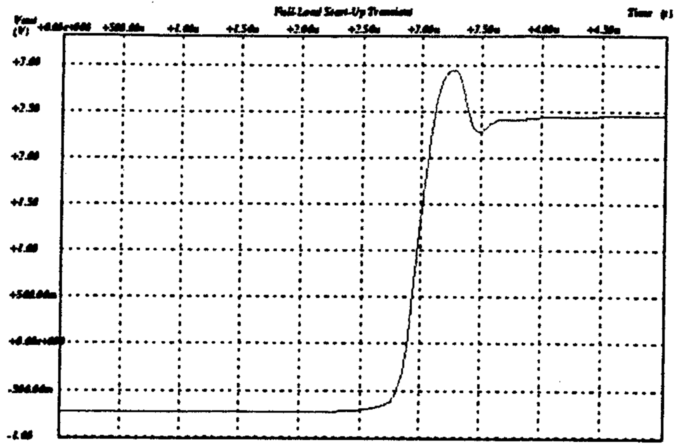


Figure 48: 5 mA circuit full-load source voltage start-up response.

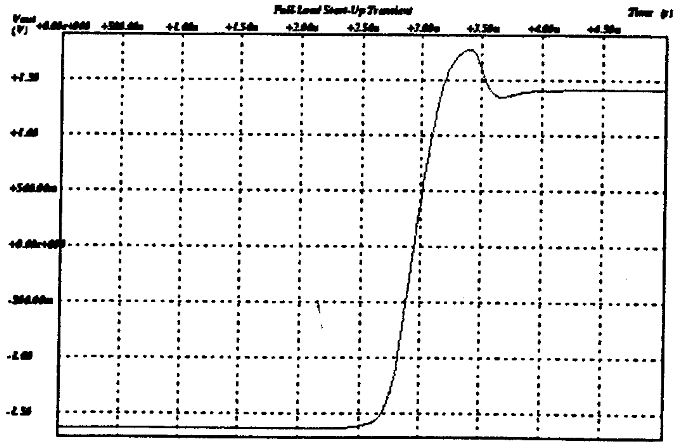


Figure 49: 500 mA circuit full-load source voltage start-up response.

4.3 Load Jump Testing

The last section of transient analysis that needs to be done is when the load jumps unexpectedly from one loading condition to another and stays there. Due to the nearly identical behavioral response between the 5 mA and 500 mA circuits, the only analysis seen in this section will be for the 500 mA circuit. There are six jumpy conditions that will be covered. They are from no-load to half-load, no-load to full-load, half-load to no-load, half-load to full-load, full-load to no-load, and full-load to half-load. Both resistive and current loads will be tested.

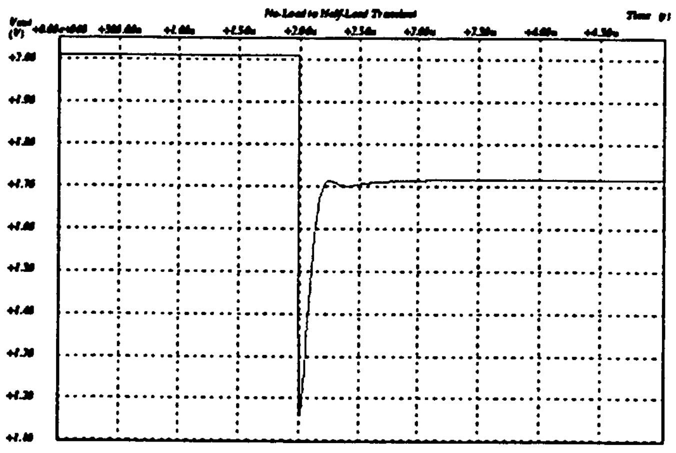


Figure 50: Resistive no-load to half-load transient.

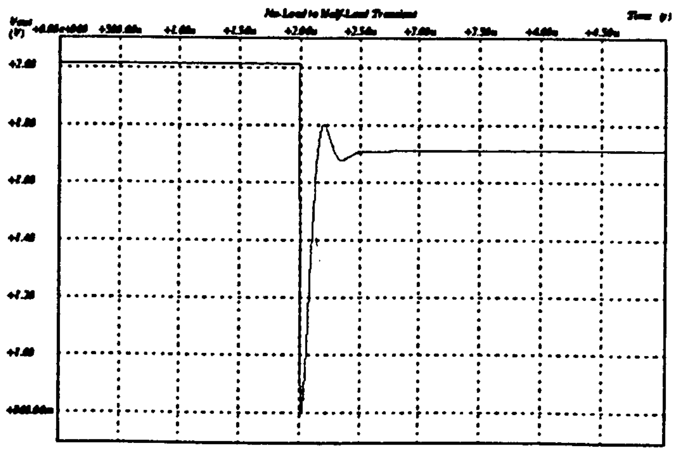


Figure 51: Current no-load to half-load transient.

Looking at Figures 50 - 61, it can be noted that there is a need to somehow cut

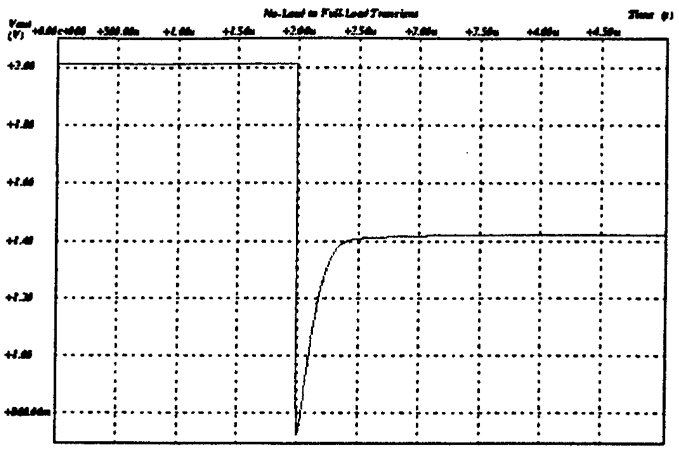


Figure 52: Resistive no-load to full-load transient.

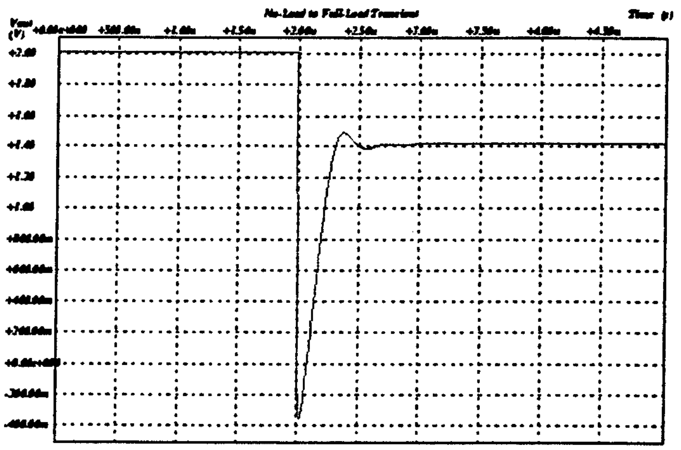


Figure 53: Current no-load to full-load transient.

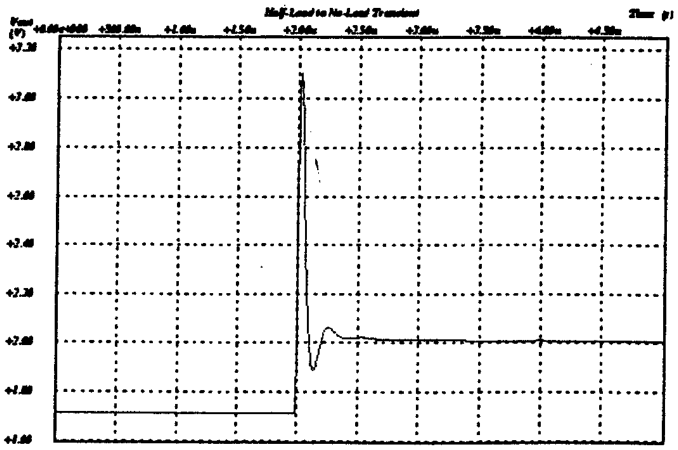


Figure 54: Resistive half-load to no-load transient.

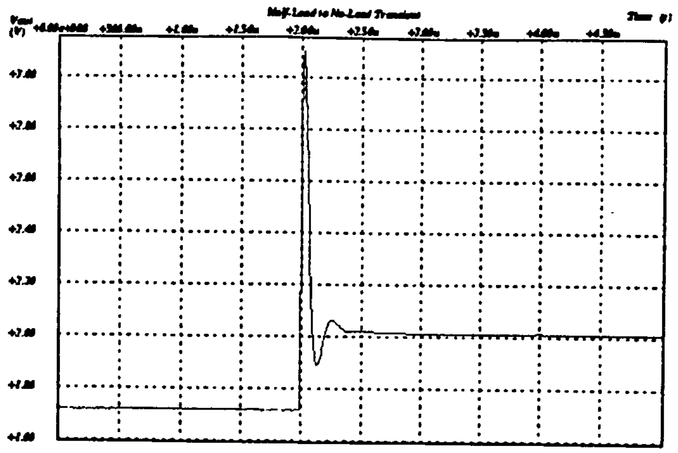


Figure 55: Current half-load to no-load transient.

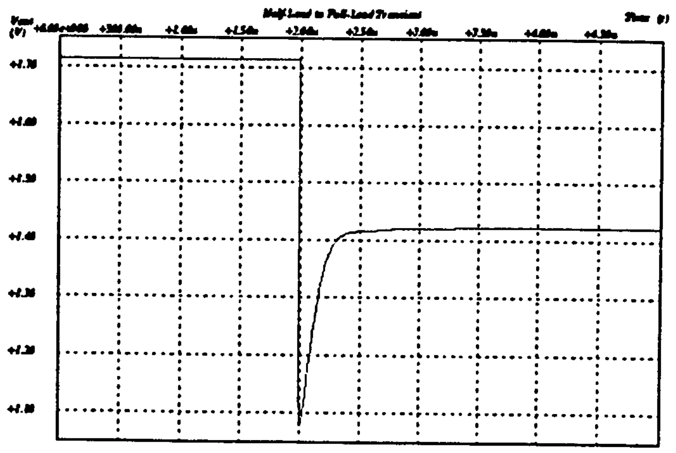


Figure 56: Resistive half-load to full-load transient.

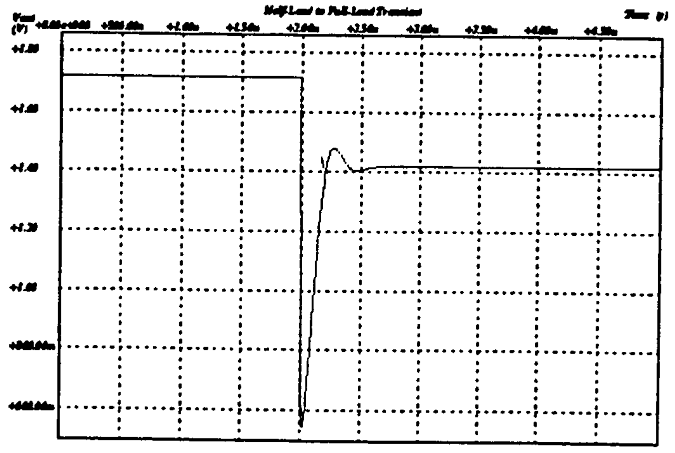


Figure 57: Current half-load to full-load transient.

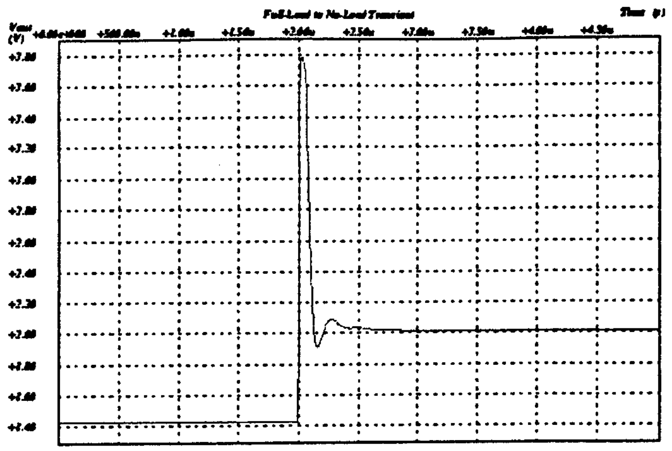


Figure 58: Resistive full-load to no-load transient.

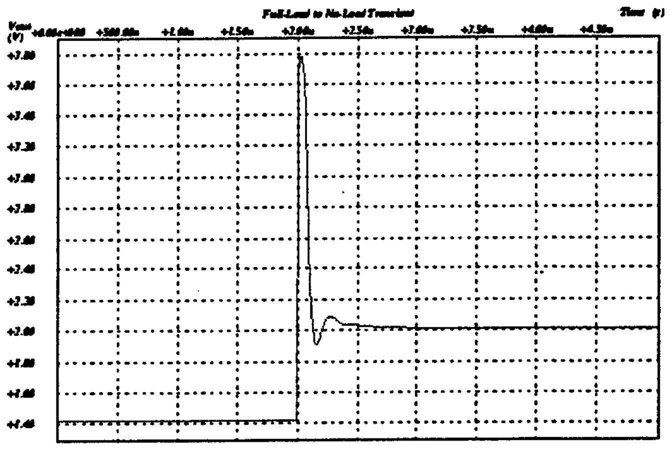


Figure 59: Current full-load to no-load transient.

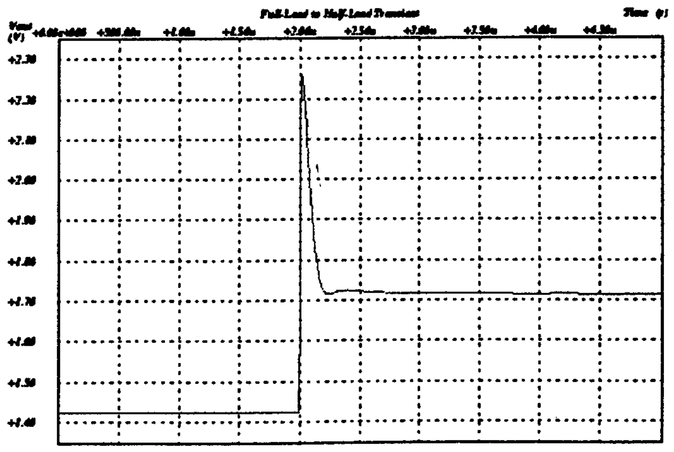


Figure 60: Resistive full-load to half-load transient.

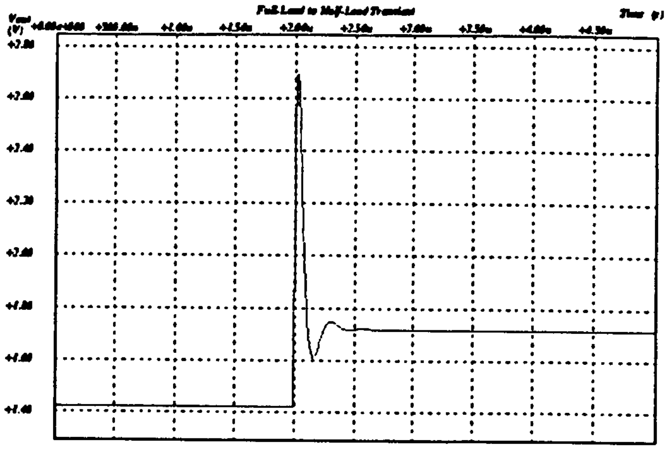


Figure 61: Current full-load to half-load transient.

down on the peak transient swings if the load looks like a current source. If the size of the peaks were to somehow diminish, the time to steady-state operation would be quicker. This means that the equipment will have a longer operational life. This topic is discussed in Chapter 5.

5 Dynamic Operation

When looking at *MEMs*, the first impression reached is that due to the devices negation of both gravity and momentum, they are going to be really fast switching devices. This poses a problem in todays technology due to the The best speed obtained so far in this thesis is around 5 MHz.

Better ways of implementing circuits must still be devised and implemented for improvement to occur. In this third portion of the thesis, two ways were devised. If you look in a hierarchal way, one exists within the other. The first way depicted in Figure 62 shows a pseudo-dynamic configuration that takes in signals from off the chip to change the beta network and thus change the gain of the circuit. In this way,

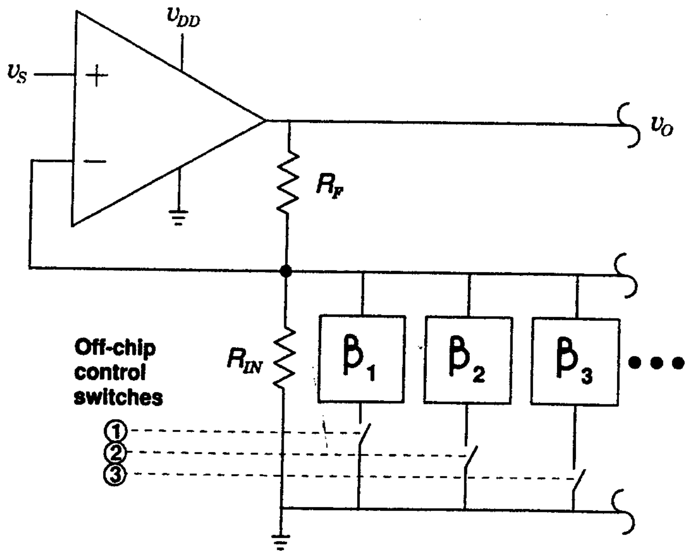


Figure 62: User configurable power supply.

the circuit is dynamic since the user who down-loads this power supply can change the output voltage of the circuit by changing the beta network. To do this, the network has to be set up using FPGAs[5]. FPGAs stands for field programmable gate arrays. Basically they are a bunch of re-programmable transistors fabricated on a chip that the user can configure in any way deemed plausible. Once the user is done with that use, the chip can be reconfigured for the next project and so forth.

The second dynamic way devised provides a method to limit the overshoot of the output voltages during transient operations. The concept is for the circuit to change with regard to the output voltage transients to help keep itself closer to the target voltage. This is laid out in Figure 63. The objective of this method is to develop a

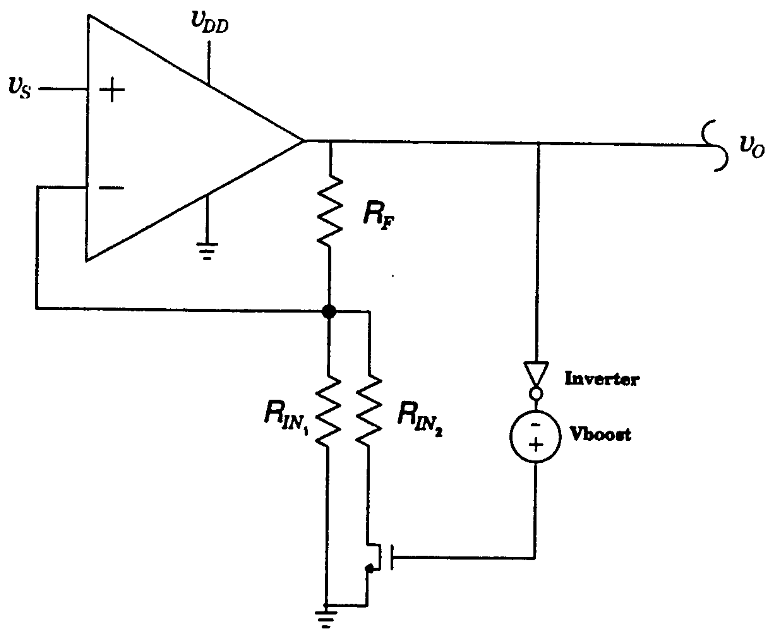


Figure 63: Proposed dynamic feedback system.

beta network that changes when the output voltage changes. For this to happen, what is needed is a network that is inversely proportional to the output voltage. Recall that β is proportional to R_F over R_I. Therefore what is needed is R_I increasing as V_{OUT} increases. This poses a problem in CMOS technology since the NMOS transistor in question will turn on not off. This will lead to a lower equivalent resistance and thus boosting the output voltage towards instability. What is needed is a way to negate the output voltage with respect to the controlling transistor. Depicted in Figure 64

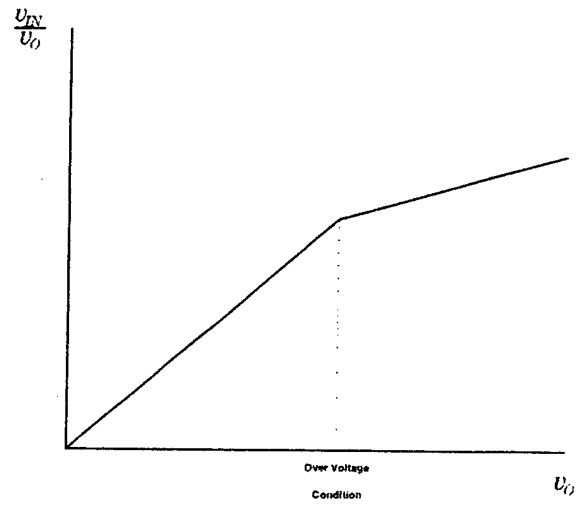


Figure 64: Theoretical dynamic feedback transfer function.

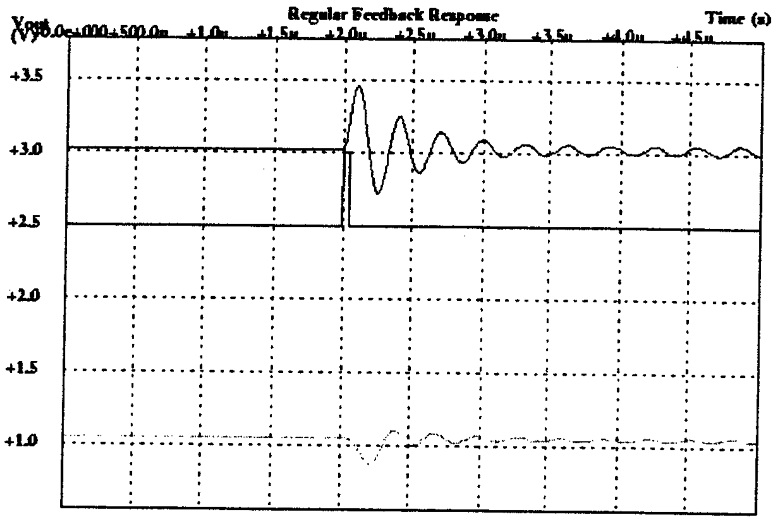


Figure 65: Circuit response with conventional feedback.

is the desired transfer function of the circuit. As output voltage increases towards an over voltage condition, the beta network reconfigures to cause more negative feedback to be developed. This will reduce the effect of overshoot oscillations. To fulfill this, one of the thesis requirements was neglected. This solution requires a 2 power rail system. With two power rails, an inverter circuit can be constructed and along with ungrounded voltage source, as output voltage increases, the beta circuit decreases and thus regulates towards the proper output voltage. Figures 65 and 66 compare the outcome of a non-dynamic feedback versus a dynamic one. . It can be seen

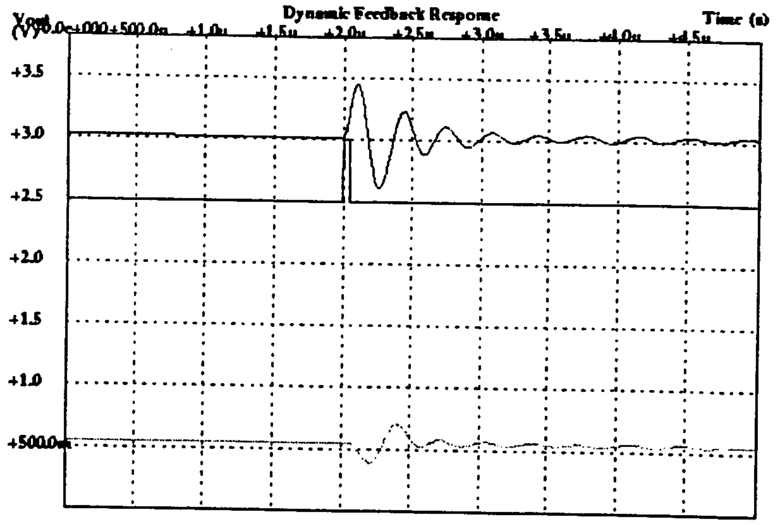


Figure 66: Improved circuit response due to dynamic feedback.

that the oscillations attenuate more quickly in the dynamic system. In this way, the voltage supplied to the *MEMs* network will be more stable. Also in these simulation results, the boost voltage that controls the dynamic feedback is seen.

6 Conclusion

Several goals were accomplished in this thesis. The first of which dealt with the design and testing of the op-amp in Chapters 2 and 3. From these chapters, the current driving capability of the op-amp attained 500 mA when a "BJT" output stage was introduced. The output stage was simulated only using SPICE. Due to this, the output produced through laying out, extracting, and then testing using SPICE might differ.

Testing both the 5 and 500 mA circuits provided evidence that they both behaved very similarly when dealing with transients. Because of this, one of the objectives stated was not dealt with in the thesis proper. The output power capabilities for both circuits are 15 mW and 1.15 W respectively. When testing these circuits under jumpy load conditions, it was a surprise to note the difference in response. An initial observation could be made that the differences in response due to the two separate ways presented for the same loading condition could be explained in terms of energy. Without further testing, no conclusions will be made at this point.

Designing for dynamic circuit behavior provided a few problems that are still not resolved. The first of which is that the fabrication of *MEMs*, FPGAs, and ASICS cannot be placed on the same chip. Several companies are heading in that direction but are not there as of yet. It has to be taken on faith that these processes will become available in the near future. When dealing with the dynamic feedback, one of the first objectives, that of one power rail, had to be discarded in order to invert the output voltage. Without this operation, this concept can not be implemented. Also, the use of the boost voltage is somewhat vague and the actual routing of voltage pins to an interior location on a chip might or might not work. Moving ahead, a problem still exists for this dynamic solution. Only one loading condition can be designed for. The obvious condition to design for is full-load since the largest signal swings would be produced.

Due to the slow nature of CMOS technology, some *MEMs* processes will not be able to use these concepts and will still have to rely on discrete power supplies. Some applications, such as bio-medical, require less than one MHz response and are ideally suited for these ideas to be realized.

6.1 Recommendation for Future Work

1. Perform detailed high-frequency analysis and therefore develop a full design process.

2. Within some VLSI tool layout, extract, and simulate capacitor, resistor, reference source, and BJT components.

3. Delve into *MEMs* and get more detailed parameters from which a more detailed project could be launched.

4. I_O and P_O capability along with scalability for libraries.

References

[1] Tanner Toolkit - T-SPICE User Guide & Reference, 1999.

[2] R. E. Siferd, *"Subsystem Design in VLSI"*, Wright State University, 1998.

[3] A. S. Sedra and K. C. Smith, *Microelectronic Circuits*, 3rd ed., New York: Saunders College, 1991.

[4] P. E. Allen and D. R. Holberg, *CMOS Analog Circuit Design*, New York: Oxford, 1987.

[5] Altera, *Altera Data Book*, 1998

<u>*TASK 7*</u>

Parameterized Symbolic Matrix Performance Modeling Methodology for Operational Amplifiers

Principal Investigator:	Prof. Joseph Nevin
Institution:	University of Cincinnati, Cincinnati, Ohio
Period of Performance:	March 24, 1998 to May 31, 1999
Task:	The contractor shall develop a parameterized symbolic matrix performance modeling methodology for a low-power CMOS operational amplifier using SPICE's internal matrix representation.

Modular Macromodel of an Operational Amplifier

Lakshminarayanan Venkatachalam and Joseph Nevin
Electrical and Computer Engineering and Computer Science Department
University of Cincinnati

Abstract

We describe a macromodeling method particularly applicable to operation amplifiers. Based upon an approach using SPICE by Brinson and Faulkner [1], the model is efficient, robust, and accurate. Rather than use SPICE, we elected to use the new IEEE mixed-signal modeling and simulation language VHDL-AMS[2][3] to experimentally verify the macromodeling approach.

1 Introduction

Macromodeling of electronic circuits is desireable particularly if sufficient behavioral accuracy can be obtained when the model is simulated. With the recent availability of mixed-signal description languages and simulators like VHDL-AMS, macromodeling becomes imporatant to allow rapid simulation of large systems where many individual analog circuits are involved. We present a macromodeling approach for a common component in analog circuits, operational amplifiers (opamps), that has been demonstrated to be both simple and accurate.
The macromodel described here follows a structured approach. The model contains three essential core modules (input stage, gain stage and output stage). The model also contains a number of optional modules which simulate specific parameters. Further, modules can be added to simulate other parameters not included here.
Manufacturers usually do not simulate using a device-level model, but use macro modeling to represent the behavior as seen from the input/output terminals of the circuit. Most opamp models are based on the Boyle macromodel and developments from it. The major drawback of this model is that the derivation of component values is not straightford. Some parameters are modeled using unbalanced input devices, while other parameters interact. The use of a structured modulular design approach for developing macromodels enables individual parameters to be modeled separately. The various modules can be combined to produce the output. Since the various parameters are independant, they do not interact and only those required need be included. The component values can be derived directly from the published data sheets.
The block diagram for the modular macromodel is shown in Figure 1. The various blocks represent specific amplifier performance features modeled by electrical networks. The values of the network components are calculated directly from the manufacturer's data sheet as will be shown later. Each network consists of one or more components which model a single amplifier parameter. This approach ensures that changes to one particular amplifier parameter do not indirectly modify other modeled parameters. Local voltage scalings employed in some of the blocks are not propagated outside individual blocks. One or more blocks can be added or removed from the model without affecting the operation of the remaining blocks. The input, gain and output stages are a minimum requirement for the model to perform as an amplifier. The other stages are optional and their inclusion is dictated by the application. For example, the voltage and current limiter stages can be removed when simulating a circuit for small signal AC performance.

1.1 Macromodel Modules

This section describes the various modules that make up the macromodel for an opamp.

1.1.1 Linear Modules

The linear modules included here are input stage (INPSTG) shown in Figure 2, gain and dominant pole stage (POLE1) shown in Figure 3, output stage (OUTSTG) shown in Figure 4, second pole stage (POLE2) shown in Figure 5, common mode stage (CMZERO) shown in Figure 6, and the summing stage (VSUM) shown in Figure 7. Of these, the INPSTG, POLE1 and OUTSTG form the core stages.

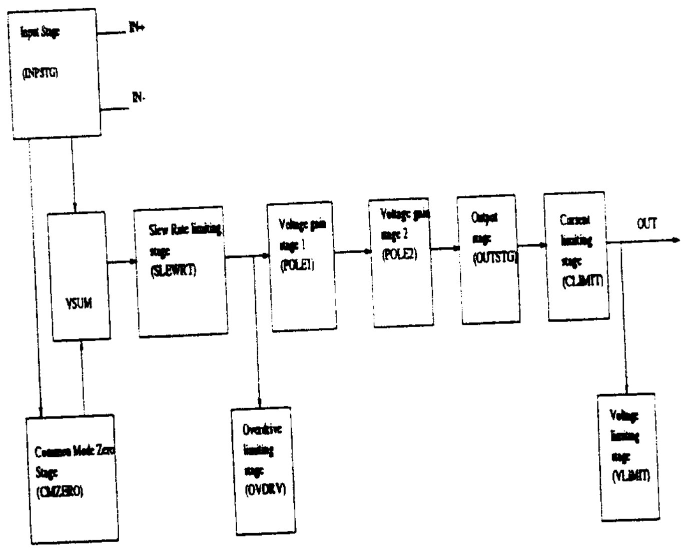


Figure 1: Block Diagram of Modular Macromodel of an Operational Amplifier

The INPSTG includes the offset voltage Qd the bias and offset current error sources along with differential impedence components. POLE1 includes the amplifier gain and the dominant pole of the amplifier. The secondary pole is modeled in POLE2 which has a unity gain. Common-mode performance is simulated with module CMZERO. This provides the common mode gain with a zero in the response. The RC network used introduced a very high frequency pole which can be neglected. The input resistor R_{dcmz} models the common-mode input resistance. The common mode error voltage produced by CMZERO is added to the differential signal in module VSUM before the combined signal passes through POLE1.

1.1.2 Nonlinear modules

The nonlinear modules included in this model are slewrate stage(SLEWRT), current limit stage (CLIMIT), voltage limit stage (VLIMIT) and overdrive recovery stage (OVDRV).
The slew rate of the amplifier is modeled by limiting the current available to Charge C_{P1} in module POLEl. This function is provided by SLEWRT. SLEWRT is the only stage that uses internal scaling to allow for the use of higher voltages in the clamping circuit to reduce errors owing to forward bias junction voltages. Current limiting is achieved by clamping the voltage across R_{scale} with the Zener diode D_{SR} offset from ground by the fixed voltage source V_{SR1}. The output section of SLEWRT removes the internal scaling so that the module has an overall gain of unity .
Output voltage and current limiting are provided by VLIMIT and CLIMIT. VLIMIT limits the output voltage to just below the supply rails. CLIMIT uses a more,unusual configuration. The O-volt voltage source V_M senses the output current whcih controls contorlled voltage source H_{CL}. This sets the turn-on threshold for the clamping diodes D_{CL1} and D_{CL2}, which are bootstrapped to the output voltage source E_{CL}. The resulting circuit is independent of the output impedance of the macromodel.
The OVDRV stage serves to model the amplifier's recovery time from voltage overdrive. This parameter is not usually modeled in the conventional models. This stage clamps the output of

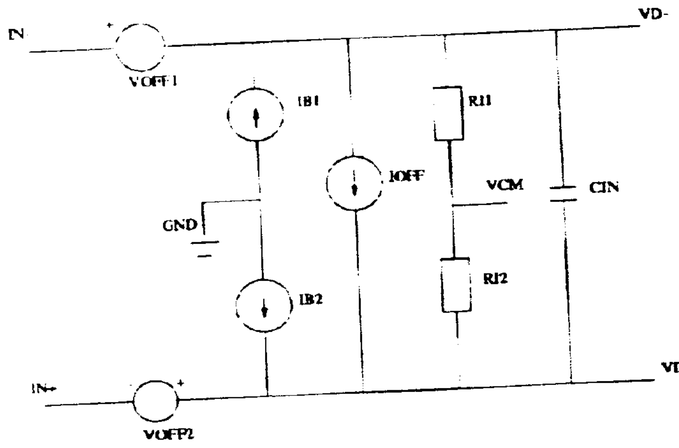


Figure 2: INPSTG Stage

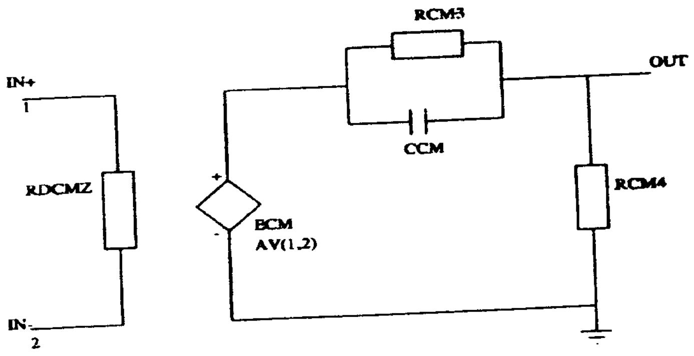


Figure 3: CMZERO Stage

the SLEWRT module at a level close to the supply voltages. In conjunction with the action ofthe VLIMIT module at the output, this has the effect of delaying the restoration of linear circuit behavior following the overload.

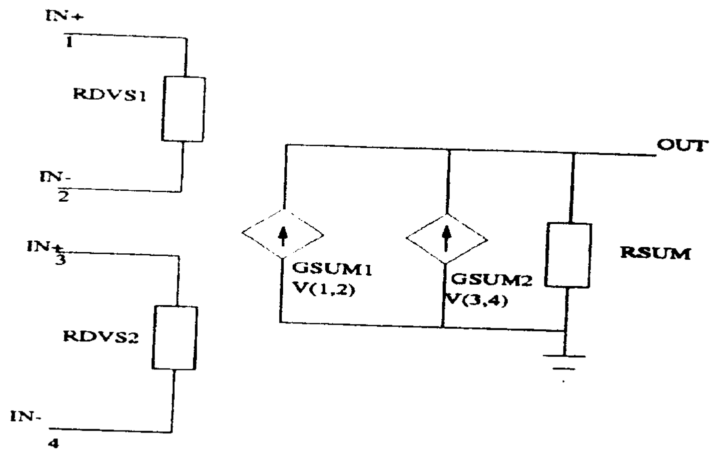


Figure 4: VSUM Stage

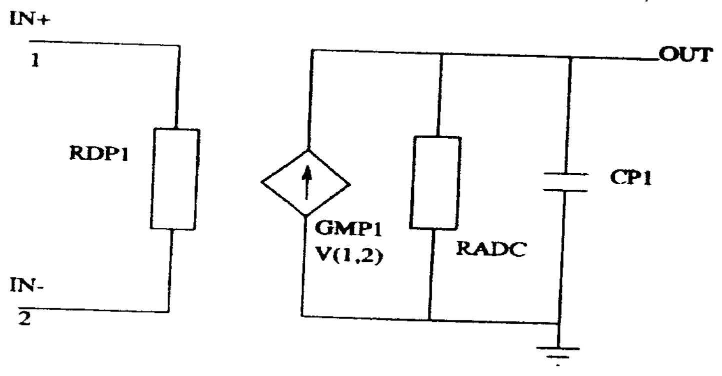


Figure 5: POLE1 Stage

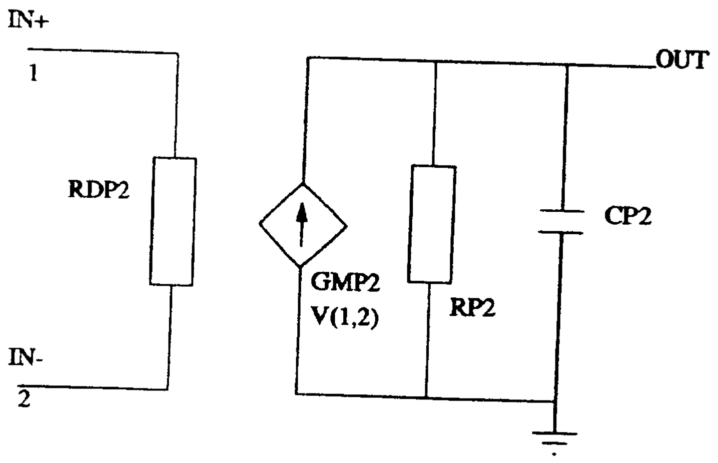


Figure 6: POLE2 Stage

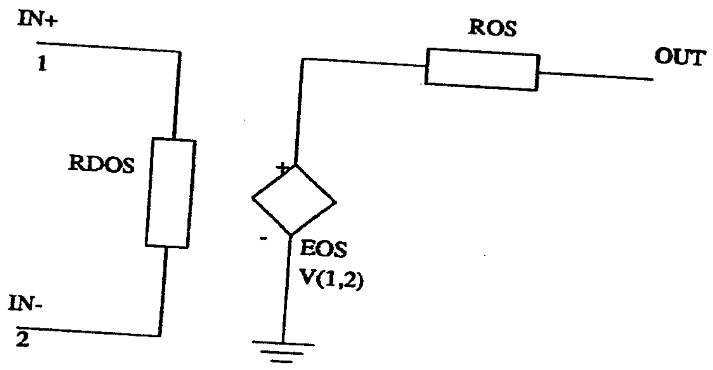


Figure 7: OUTSTG Stage

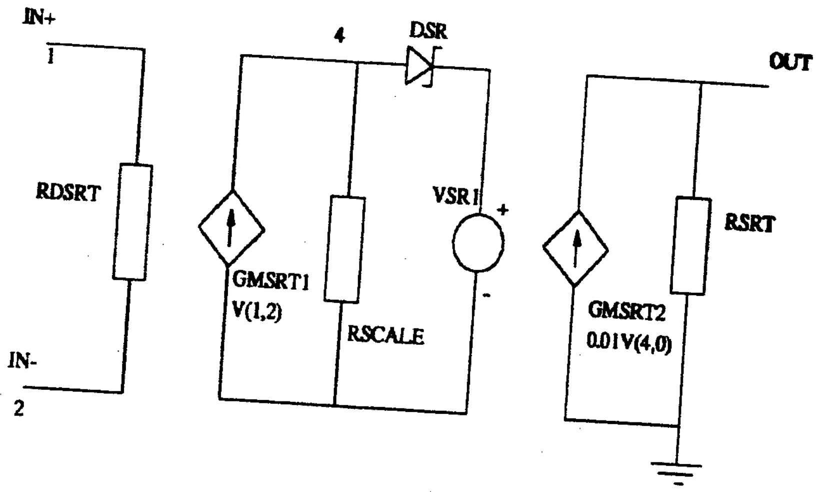


Figure 8: SLEWRT Stage

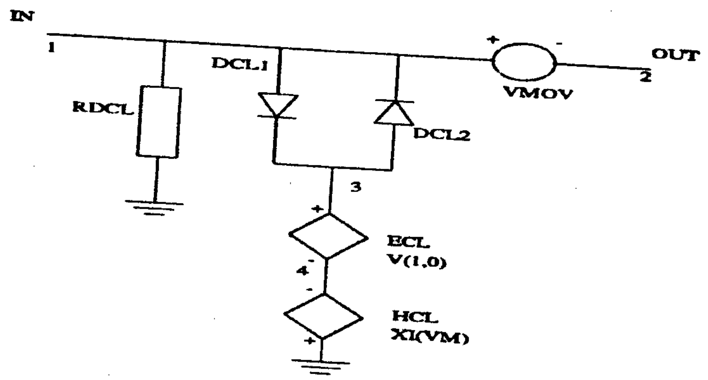


Figure 9: CLIMIT Stage

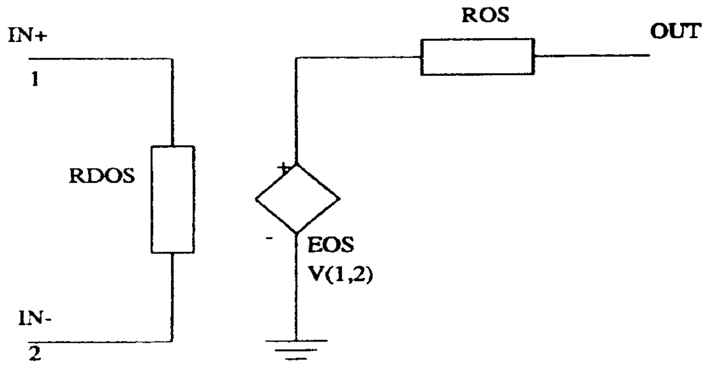


Figure 7: OUTSTG Stage

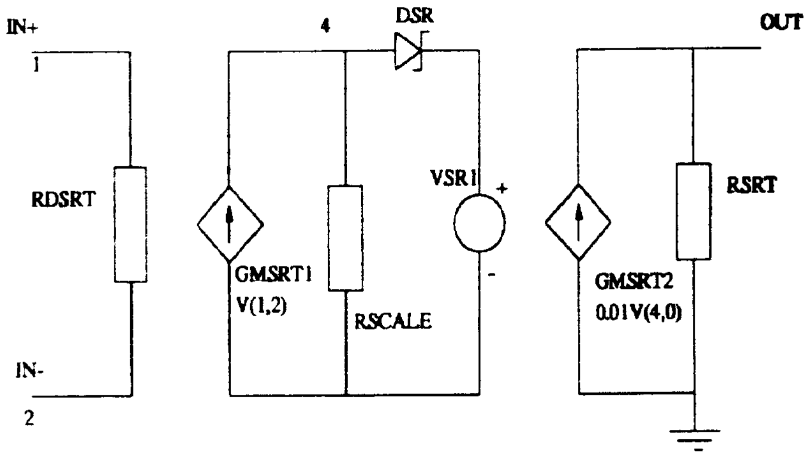


Figure 8: SLEWRT Stage

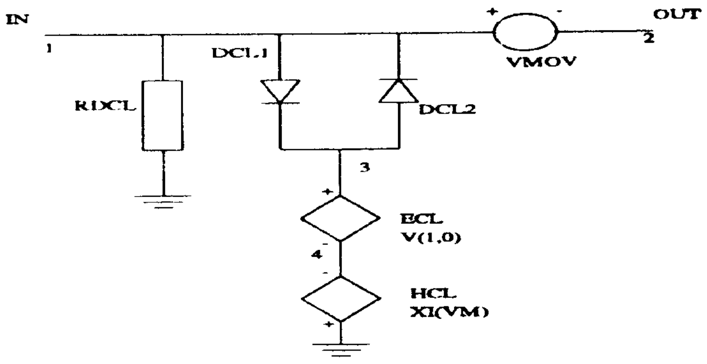


Figure 9: CLIMIT Stage

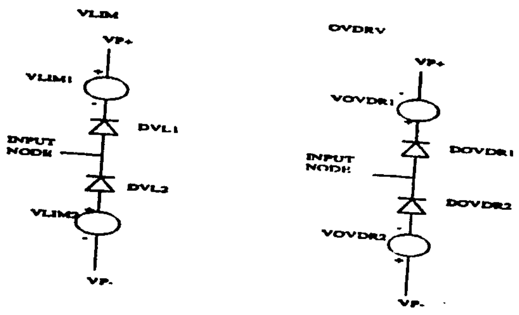


Figure 10: OVDRV and VLIMIT Stages

2 Results for a A741 OpAmp

The A741 OpAmp is one of the most common operational amplifiers in use today. Using ‹ approach, we simulate the operation of a A741 and compare the results with equivalent SPI‹ simulations simulations. All simulations were run on a SUN 4 Enterprise Server with four Spa processors. The operating system was SUNOS 5.7. SPICE 3 was used for all SPICE simulatio SEAMS 1.1c [3] was used to perform all VHDL-AMS simulations.

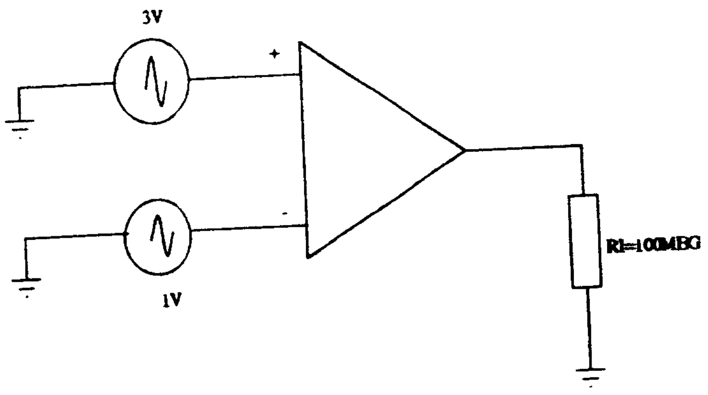


Figure 11: Operational Amplifier in Open Loop

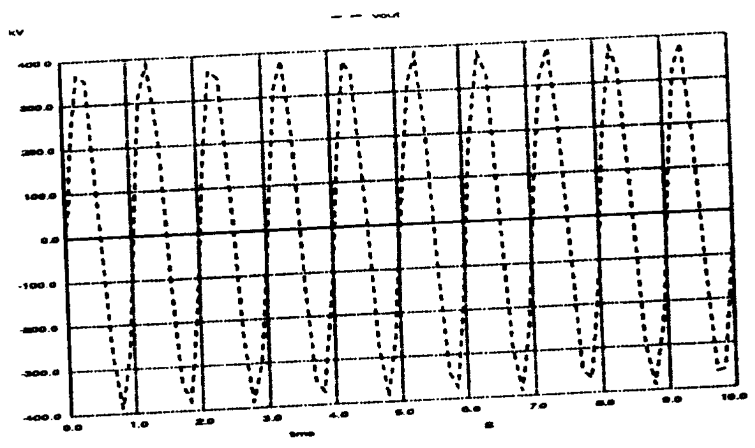


Figure 12: Spice output for a uA741 connected in open loop with 1 Mz input

2.1 741 OpAmp in Open Loop

The circuit topology used for testing the open loop response of a A741 is shown in Figure 11. The inputs are -3V to 3V, and -1V to 1V sinewaves at the non-inverting and inverting terminals respectively.The input signals are in phase. The output is taken at the load resistor with a value of 1OO Mohms. The simulation results from SPICE are shown in Figure 12, and the VHDL-AMS simulation results for the equivalent model are shown in Figure 13. The frequency of the input is 1Hz. It can be seen that the results are very close to one another with the error between the two simulation results being less than 1. Significant frequency effects are seen when the input is increased to 1-MHz as shown in the two simulator results, Figure 14 (SPICE) and 15 (VHDL-AMS). It should be noted that the results indicated here are without the voltage limiting stage in place. With the voltage limiting stage, the output saturates at the desired voltage 13V as specified for A741 OpAmps.

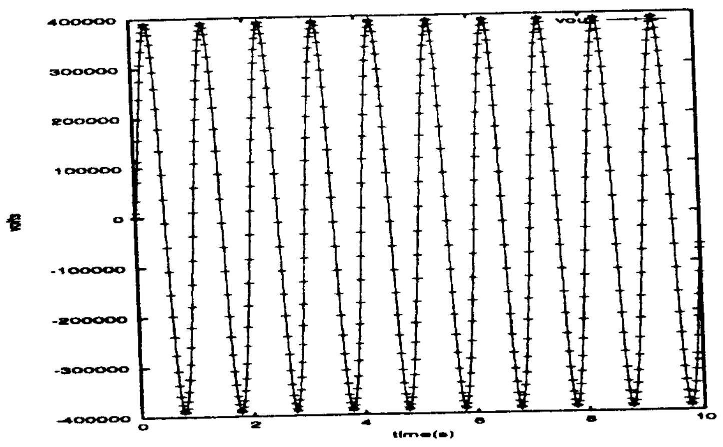


Figure 13: VHDL-AMS output for a μA741 connected in open loop with input frequency of 1 Hz

The VHDL-AMS and SPICE response for the opamp in open loop mode at high frequency show some differences. The SPICE result settles down to its final value earlier than the VHDL-AMS response indicating a difference in the DC constants between the two responses. This is due to the fact that the component models in SPICE are described with more precision than for the VHDL-AMS models. Also the fact that the VHDL-AMS model uses a static diode model contributes to this difference. It is interesting to note that the closed loop response does not show this difference, as will be shown later.
The voltage limiting stage limits the output voltage to just below the supply voltage. The results obtained while simulating the model with the voltage limiting stage in place in SPICE and VHDL-AMS are very close as seen in Figures 16 and 17. SPICE output for a A741 connected in open loop as in Figure11 with the voltage limiting stage
The output from SPICE simulation and VHDL-AMS for an input square wave is shown in Figure 18 and Figure 19. As with the open loop response with sine wave input, we notice a difference in the two results. This difference can also be attributed to the difference in the component models. The output of the opamp at higher input frequencies is shown in Figure 20 and 21. Again the outputs obtained from both SPICE and VHDL-AMS simulations are very similar. The error between the two results is about O.O9 and is well within the acceptable range.

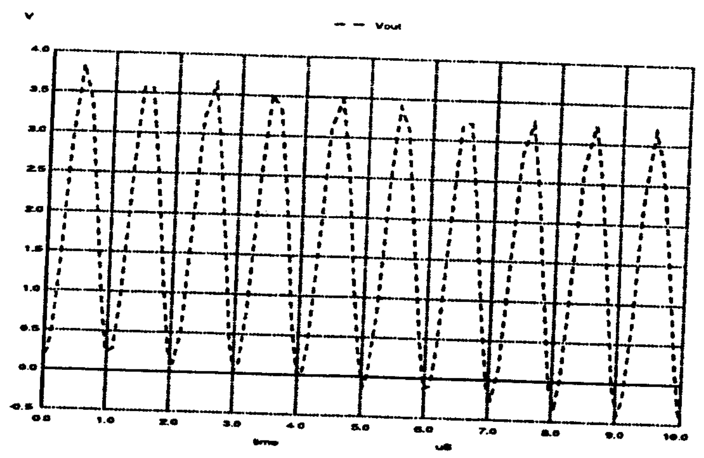


Figure 14: Spice output for a μA741 connected in open loop with 1 MHz input

2.2 Closed Loop Response of A741 OpAmp

We compared the response of the model connected as a voltage follower as shown in Figure 22. The output of the opamp is fed back to the inverting terminal. The input is a -3V to 3V sinewave and is fed to the non-inverting terminal of the opamp. The gain when connected as described above is unity.

The responses of this circuit in SPICE and VHDL-AMS for a very low input frequency are shown in Figures 23 and 24. The slewing at higher frequencies is illustrated in Figure 25 and 26, respectively. The response of the circuit shown in Figure 22 when the input is a OV to 3V square wave at 1 Hz is shown in Figures 27 and 28. Again, the results in VHDL-AMS and SPICE compare favorably with the error being less than 1.

2.3 Transient Response for Opamp with Capacitive Load

The circuit used to examine the transient response of the opamp with a capacitive load is shown in Figure 29. The result obtained from the VHDL-AMS simulation is identical to the transient response of the A741 as shown in the data sheets. The transient response obtained using VHDL-AMS is shown in Figure 30. The response is identical to the one shown in the data sheet for a A741. The error is about O.4

2.4 Common Mode Rejection Ratio

In the absence of input offset, an opamp should respond only to the voltage difference between its inputs, or $V_0 = G(V_P - V_N)$. A practical opamp is somewhat sensitive to the common mode input voltage $V_{CM} = (V_P + V_N)/2$. Its transfer characteristic is thus $V_0 = G(V_P - V_N) + a_{MV}V_{CM}$, where a is the differential mode gain, and a_{CM} is the common mode gain. The CMRR vs. fequency curve for a μA741, obtained using VHDL-AMS, is shown in Figure ??.

3 Conclusions

To our knowledge, this is the first attempt to behaviorally model an operational amplifier using the mixed-signal langauge VHDL-AMS. The flexibility of VHDL-AMS to permit modeling at various

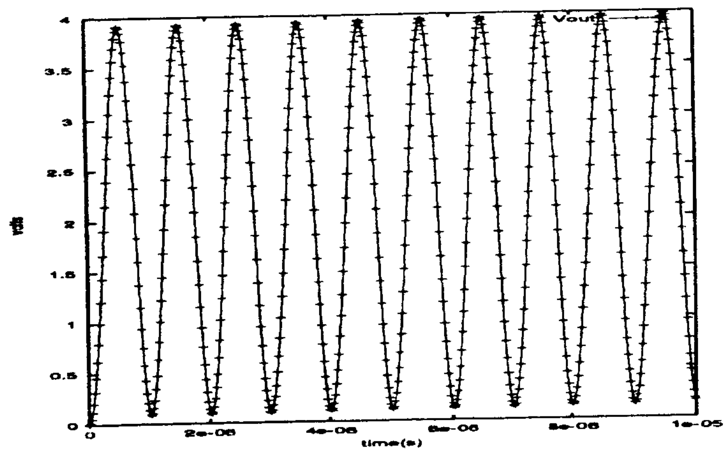


Figure 15: VHDL-AMS output for a μA741 connected in open loop with 1 MHz input

user-defined levels of abstraction led to a simple model that performed well, at least as measured against equivalent SPICE simulations.

References

[1] ACUNA, E. L., DERVENIS, J. P., PAGONES, A. J., YANG, F. L., AND SALEH, R. A. Simulation techniques for mixed analog/digital circiots. *IEEE Journal of Solid-State Circuits 25*, 2 (April, 1990), 353-363.

[2] IEEE*Standard VHDL Language Reference Manual (Integrated with VHDL-AMS changes*, 1998.

[3] FREY, P., NELLAYAPPAN, K., MAYILADUTHURAI, R. S., CHANDRASHEKAR, C. L., CARTER, H. W. SEAMS: Simulation Environment for VHDL-AMS. In *Proceedings of the 1998 Winter Simulation Conference.* (1998), pp. 539-546.

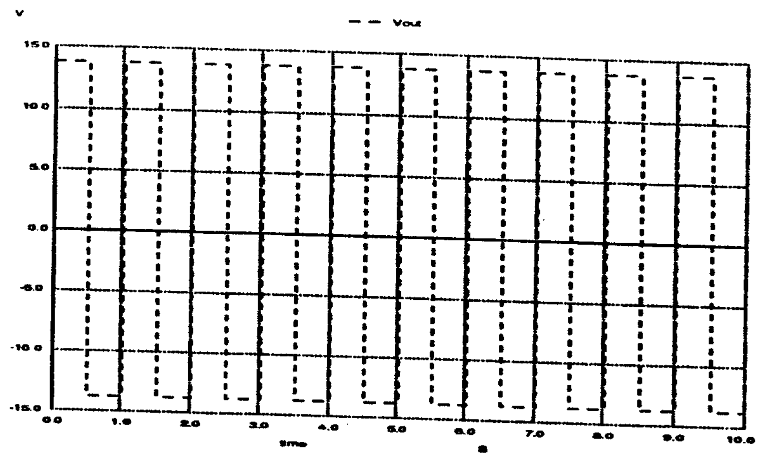


Figure 16: Spice output for a μA741 connected in open loop with voltage limiting stage

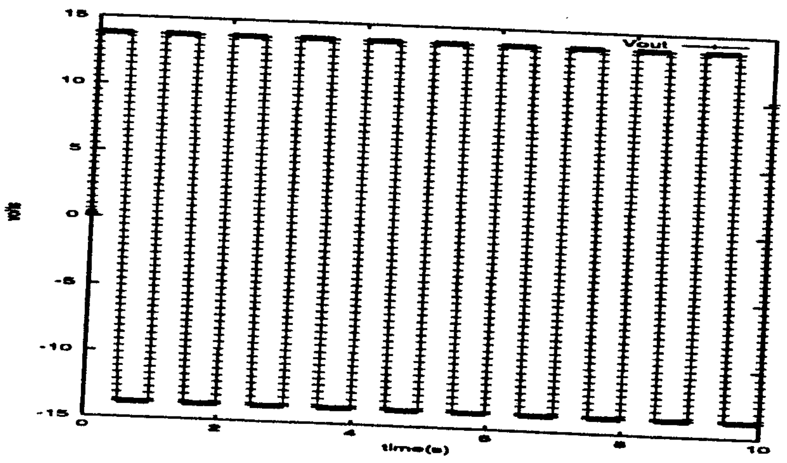


Figure 17: VHDL-AMS output for a μA741 connected in open loop with voltage limiting stage

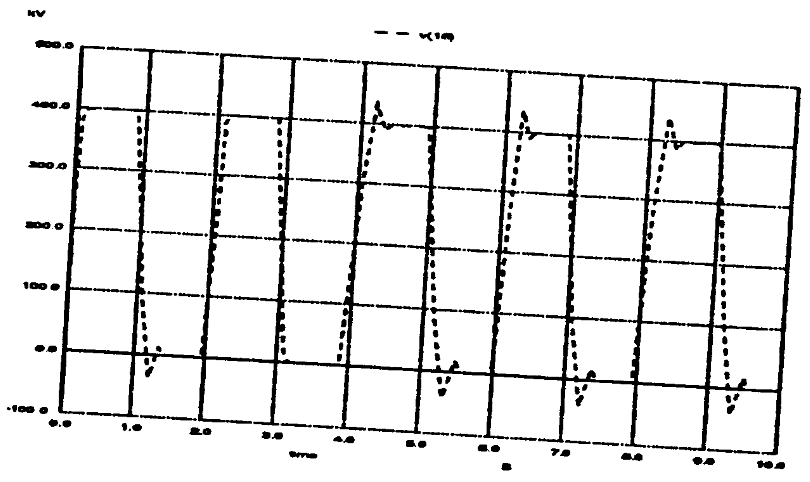


Figure 18: Spice output for a μA741 connected in open loop with 1 Hz square wave input

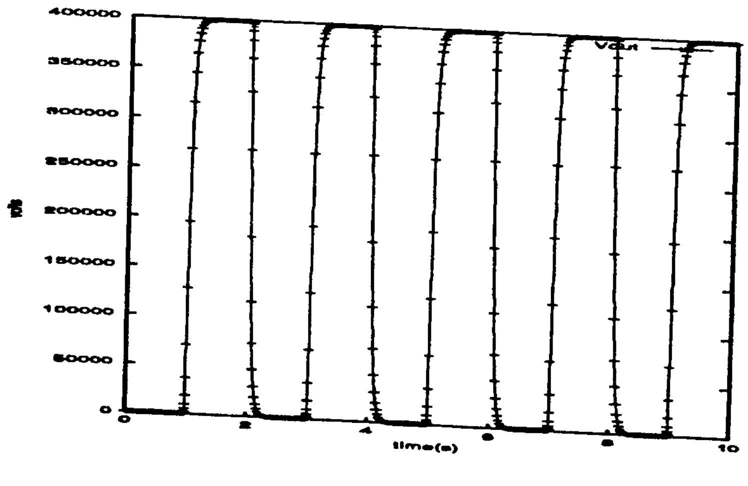


Figure 19: VHDL-AMS output for a μA741 connected in open loop with 1 Hz square wave input

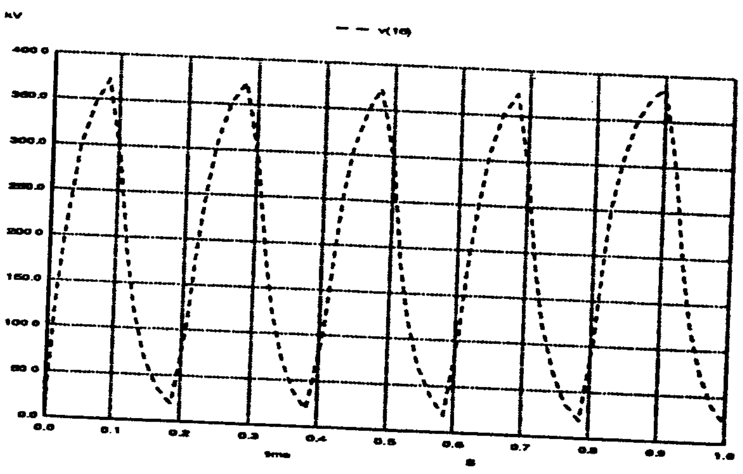


Figure 20: Spice output for a μA741 connected in open loop with 10 Hz square wave input

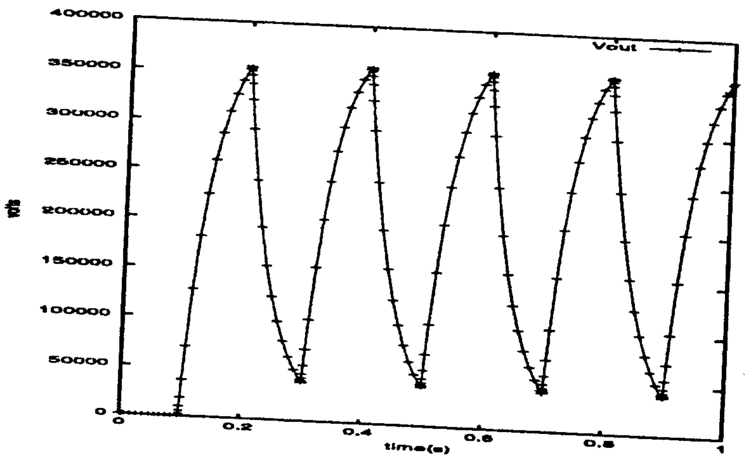


Figure 21: VHDL-AMS output for a A741 connected in open loop 10 Hz square wave input

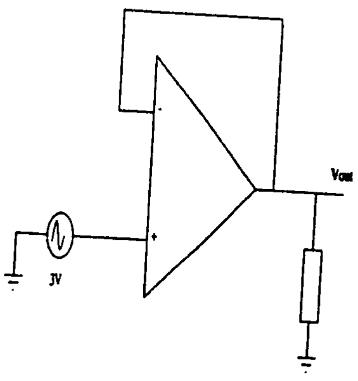


Figure 22: Opamp connected as a voltage follower

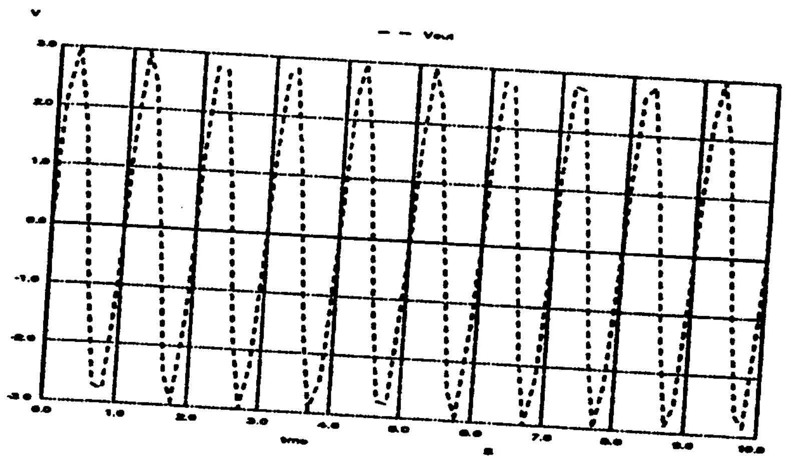


Figure 23: Spice output for a A741 in voltage follower mode with 1 Hz sine wave input

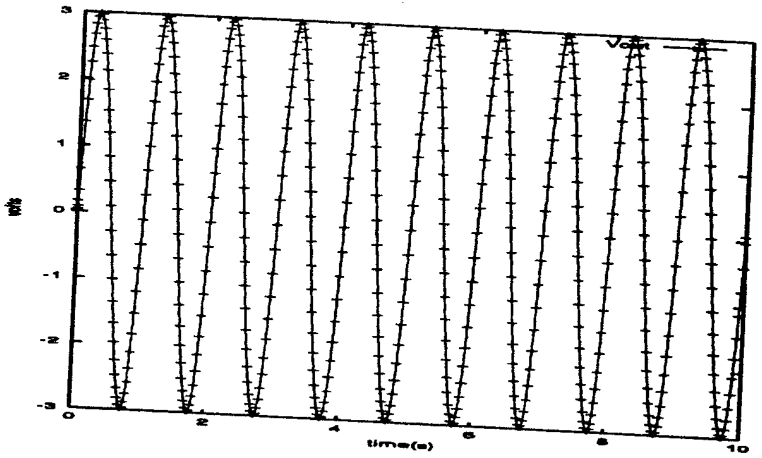


Figure 24: VHDL-AMS output for a A741 in voltage follower mode with 1 Hz sine wave input

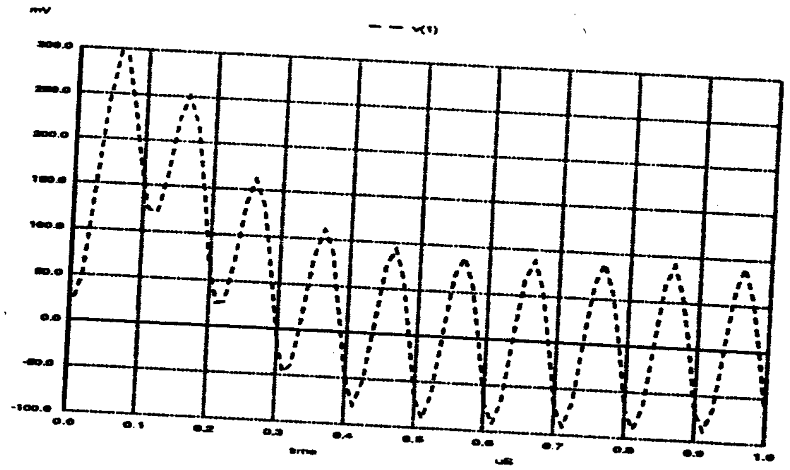


Figure 25: Spice output for a A741 in voltage follower mode with 10 Hz sine wave input

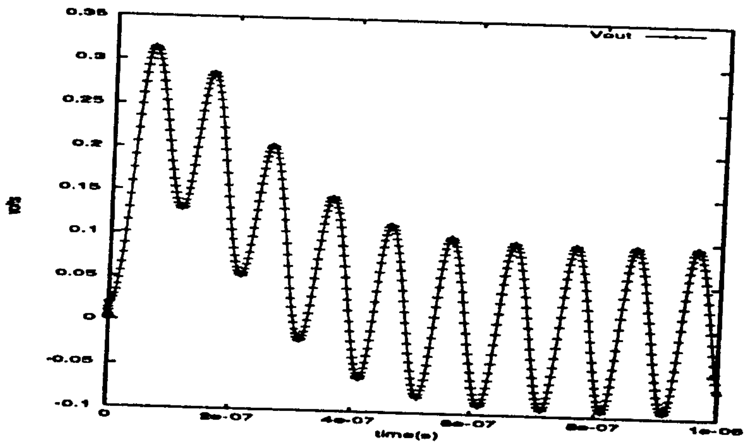


Figure 26: VHDL-AMS output for a A741 in voltage follower mode with 10 Hz sine wave input

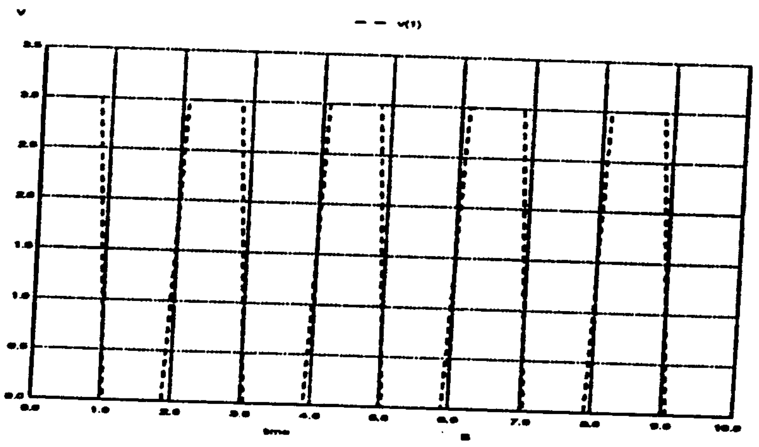


Figure 27: Spice output for a A741 in voltage follower mode with 1 Hz square wave input

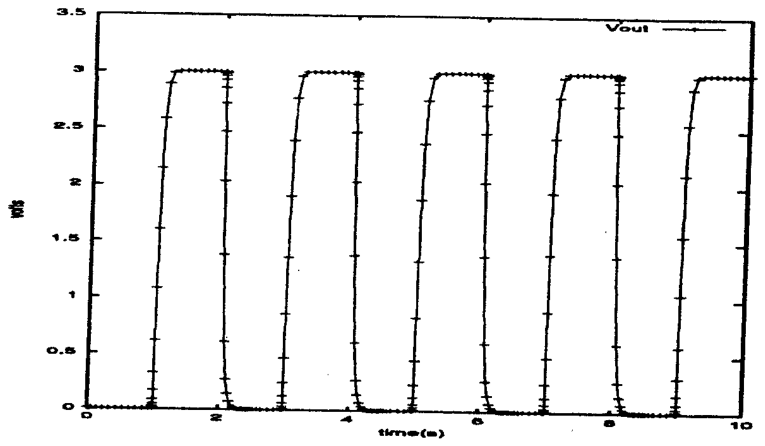


Figure 28: VHDL-AMS output for a A741 in voltage follower mode with 1 Hz square wave input

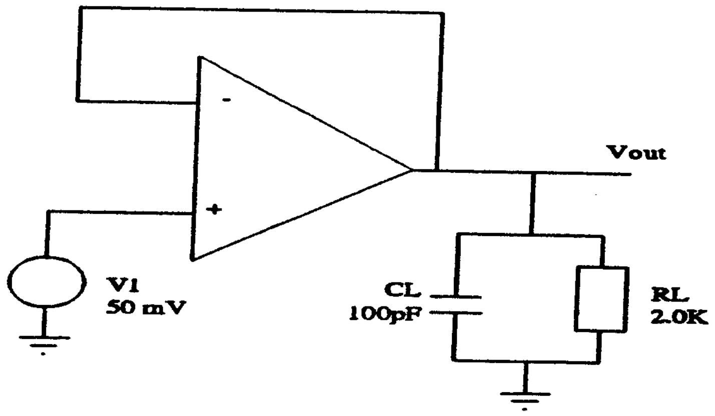


Figure 29: Circuit to obtain transient response of operational amplifier

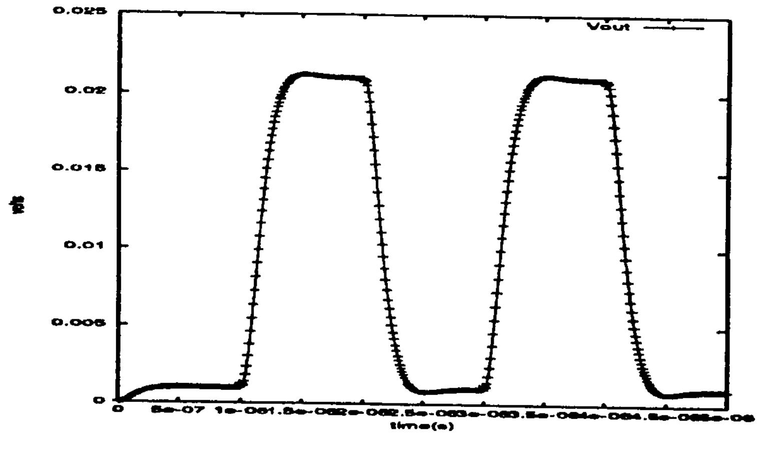


Figure 30: Transient response of μA741 operational amplifier

TASK 8

Matrix-Based Database for Symbolic Analysis

Principal Investigator:	Prof. Carla Purdy
Institution:	University of Cincinnati, Cincinnati, Ohio
Period of Performance:	March 24, 1998 to May 31, 1999
Task:	The contractor shall develop an integrated control and management tool for dealing with system-level development. The systems to be analyzed contain traditional VHDL subsystems which are integrated into microeclectromechanical (MEMS) devices. Such systems are part of an emerging technology collectively known as mechatronics. This technology seeks to manage all aspects of complex electromechanical systems in an efficient and cost-effective manner.

ع

Matrix-based Database for Symbolic Analysis and Simulation

Dennis Gib son and Carla Purdy
University of Cincinnati

March 1999

In this task we examined and compared a variety of simulation tools for use in MEMS simulation in particular. We found that agreement among simulation results was generally good but that more development of symbolic tools is needed to increase their efficiency if they are to compete with current numerical methods. Results are summarized below and have also been included in [Gibson, 1999, 2002]:

1. Simulation Tools Examined

1.1. SPICE

SPICE is a general-purpose electric-circuit simulation program. The name stands for Simulation Program with Integrated Circuit Emphasis. The allowed components are resistors, capacitors, inductors, mutual inductances, independent dc and ac sources, dependent sources, transmission lines, diodes, and transistors [Tuinenga, 1992].

SPICE is capable of performing three main types of analysis. It can determine the dc behavior of selected output voltages with respect to changes in input voltages. A second type of analysis that is usually required in order to fully determine a circuit's behavior is called a transient analysis. Transient analyses calculate circuit voltages and currents with respect to time. This assumes that there is a time-dependent object that causes an effect on the rest of the circuit. The third type of analysis that SPICE can perform is called an ac analysis. This type is also referred to as a sinusoidal steady-state analysis. Voltages and currents are calculated as a function of frequency. In an ac analysis output variable changes are calculated in response to changes in the amplitude, frequency or phase of sinusoidal input voltage or current sources.

1.2. SEAMS

SEAMS stands for *Simulation Environment for VHDL-AMS.* This is an analog and mixed signal simulator. The systems takes as an input, a VHDL/VHDL-AMS description and goes through the following stages : parsing, elaboration, code generation and simulation. The entire system rests on a Time Warp system developed here at the University of Cincinnati. Optimistic Synchronisation Protocols [Frey, 1998], are used to implement this mixed signal simulator.

VHDL is the VHSIC (Very High Speed Integrated Circuit) Hardware Descriptive Language. VHDL is an international standard specification language for describing digital hardware used by industry worldwide. VHDL enables hardware modeling from the gate to system level. VHDL provides a mechanism for digital design and reusable design documentation. VHDL is the outcome of a US Government request for a new means of describing digital hardware. The need for a common language to describe and communicate digital design was clear [Ashenden]. VHDL has numerous advantages [VHDL]:

1) Design Methodology: VHDL supports many different design methodologies (top-down, bottom-up, delay of detail) and is very flexible in its approach to describing hardware.

2) Technology Independence: VHDL is independent of any specific technology or process. However, VHDL code can be written and then targeted at many different technologies.

3) Wide Range of Descriptions: VHDL can model hardware from a high level to a low level. VHDL can describe hardware from the standpoint of a "black box" to the gate level. VHDL also allows for different descriptions of the same component and allows the designer to mix behavioral descriptions with gate level descriptions.

4) Standard Language: The use of a standard language allows for easier documentation and the ability to run the same code in a variety of environments. Communication among designers and among design tools is enhanced by a standard language as well.

5) Design Management: Use of VHDL constructs, such as packages and libraries, means that common elements can be shared among members of a design group.

6) Flexible Design: VHDL can be used to model digital hardware, and other types of systems.

The IEEE standardized VHDL and the current standard is VHDL 1076-1993 [Standards, 1997]. VHDL-AMS (where the AMS stands for Analog Mixed Signal), is an effort to standardize an extension of VHDL 1076 to support the description and the simulation of analog and mixed-signal circuits and systems. Formally, VHDL-AMS is known as VHDL 1076.1. VHDL-AMS is a strict superset of VHDL 1076-1993.

Where VHDL 1076 deals with the discrete domain, VHDL-AMS extends to the continuous domain. Since VHDL-AMS continuous models are based on differential algebraic equations (DAE's), then any domain that can be expressed with DAE's can be modeled in VHDL-AMS. The differential equations which describe state variable solutions of systems in the mechanical, thermal, etc. disciplines, are of the same form as the differential equations in the electrical discipline.

1.3. ANSYS

The response of most real-world engineering systems to applied actions is usually difficult, if not impossible, to determine by a closed-form mathematical solution. The finite element method offers a convenient way of obtaining approximate solutions to just about any engineering problem.

The name *finite element* summarizes the basic concept of the method: the transformation of an engineering system with an *infinite* number of unknowns (the response at every location in a system) to one that has a finite number of unknowns related to each other by elements of finite sizes.

The unknowns, called degrees of freedom, represent the responses to applied actions. The degrees of freedom and the actions are related by a set of basic equations. The purpose of the finite element method is to determine the solution of these equations acrosss the entire engineering system being analyzed. The simplest form of a basic equation is as follows [Ansys, 1991]:

$$[K]\{d\} = \{A\}$$

where {d} is the degree of freedom vector, {A} is the action vector, and [K] is the matrix relating {d} to {A} and is often called the stiffness or coefficient matrix. In general, [K] and {A} are known, and {d} is initially unknown.

The actual form of a basic equation is determined by the type of analysis being performed. For example, in a static structural analysis, the equation is:

$$[K]\{u\} = \{F\}$$

where [K] is the structural stiffness matrix, {u} is the displacement vector, and {F} is the force vector.

To obtain solution data for the entire system being analyzed, the [K] matrices for the individual elements are assembled into a global [K] matrix. This task is not difficult since the elements are connected to each other mathematically by their nodes. The resulting global set of simultaneous equations can then be solved for the unknowns or degrees of freedom.

ANSYS uses a frontal (wavefront) equation solver which performs the asembly and solution steps in parallel. Once the degrees of freedom are determined, derived results are calculated within each element using its shape functions. Stress and strain would be examples of derived results for a structural element.

ANSYS allows input from files, which allows for the creation of a template input file in which parameters for our cantilever beams can be introduced.

2. Extending VHDL-AMS to Finite Element Analysis

2.1. Introduction

Since VHDL-AMS is an ordinary differential equation solver, it is ideal for solving systems of equations. We demonstrate its effectiveness and ease by modeling the cantilever beam (with the assumption that the beam is uniform). In this case, we only deal with a mechanical beam in which a constant load is applied. Therefore, we are examining the static behavior of the beam. Finite Element Analysis (FEA) on the beam can be done by breaking the beam into multiple elements using the equation:

$$F = KX$$

where F is the vector of forces applied to the assigned elements, K is the stiffness matrix for the beam, and X is the vector of displacements. The K matrix is actually the combination of all the stiffness matrices for each individual element of the beam. Since the endpoints of each element interact with the endpoints of an adjacent element, the K matrix takes this point into consideration. Consider a two element beam with 2 elements, A and B, where A has endpoints 1 and 2 and B has endpoints 2 and 3. A has its matrix shown as

$$K^A = \begin{bmatrix} K^A{}_{11} & K^A{}_{12} \\ K^A{}_{21} & K^A{}_{22} \end{bmatrix}$$

And B has its corresponding matrix shown as

$$K^B = \begin{bmatrix} K^B{}_{11} & K^B{}_{12} \\ K^B{}_{21} & K^B{}_{22} \end{bmatrix}$$

Therefore, the corresponding stiffness matrix for a beam with two elements will be

$$K = \begin{bmatrix} K^A{}_{11} & K^A{}_{12} & 0 \\ K^A{}_{21} & K^A{}_{22} + K^B{}_{11} & K^B{}_{12} \\ 0 & K^B{}_{21} & K^B{}_{22} \end{bmatrix}$$

2.2. Modeling Beam in VHDL-AMS with FEA

We have written a program that generates a VHDL-AMS model for a cantilever beam given the dimensions of the beam and the number of elements. The models are generated rapidly using this program. The program concentrates on filtering out useless terms, such as ignoring the 0 terms in the K matrix and using the boundary condition which assumes that the deflection at the fixed end is 0. In Figure 3.1 we show a model generated for a beam modeled with only one element.

The model in Figure 1 was reduced from a two by two matrix to just one element due to the boundary condition that deflection at Node 0 is 0. So column 1 and row 1 of the matrix are eliminated from consideration. Thus the deflection at node 1, called V1, is dependent only on the stiffness of the single element. Figure 9.2 describes the model generated for a beam partitioned into five finite elements.

```
-----------------------------------
entity FEABEAM is
end entity FEABEAM;
-----------------------------------
architecture behavior of FEABEAM is
  constant F0: real:=1.0e-5;
  constant  L: real := 80.0e-6;
  constant  W: real := 20.0e-6;
  constant  H: real := 2.0e-6;
  constant EZ: real := 170.0e9;
  constant IZ: real := (W*H*H*H)/12.0;
  constant EI: real := EZ*IZ;
  constant L3 :real := L*L*L;
  constant k : real := 3.0*EI/L3;
  constant L2 :real := L*L;
  quantity V1 : real;
begin
    V1 == F0/k;
end behavior;
-----------------------------------
```

Figure 1. FEA model of beam with one element

The model in Figure 2 results in a system of five equations and five unknowns. Thus, it can be solved quite simply. Thus, FEA may be incorporated into VHDL-AMS designs to aid in the rapid prototyping of systems. In the future, the program will be modified so that the models created will input the parameters via input file. This will aid the designer since the designer will not have to continue to recompile the models each time a set of beam parameters or different number of elements are chosen. As FEA is incorporated into VHDL-AMS, greater flexibility will be given to MEMS designers in the future.

```
------------------------------------
entity FEABEAM is
end entity FEABEAM;
------------------------------------
architecture behavior of FEABEAM is
  constant F5: real:=1.0e-5;
  constant  L: real := 80.0e-6/5.0;
  constant  W: real := 20.0e-6;
  constant  H: real := 2.0e-6;
  constant EZ: real := 170.0e9;
  constant IZ: real := (W*H*H*H)/12.0;
  constant EI: real := EZ*IZ;
  constant L3 :real := L*L*L;
  constant k : real := EI/(16.0*L3);
  constant L2 :real := L*L;

  quantity V1: real;
  quantity V2: real;
  quantity V3: real;
  quantity V4: real;
  quantity V5: real;

begin
     V1 == (2.0*V2- V3);
     V2 == 2.0*V1;
     V2 == -2.0*V3 + V4;
     V3 == 2.0*V4 - V5;
     V4 == (k*V5 - F5)/k;
end behavior;
----------------------------------
```

Figure 3.2 FEA model of 5-element beam

2.3. Advantages of this approach

The above examples illustrate how VHDL-AMS can be used for system design involving multiple energy domains. The language gives a unified approach to dealing with multiple domains. VHDL-AMS also allows for the definition of physical types. For example, "time" is a standard VHDL-AMS type. This feature facilitates understanding of domain interactions and also simplifies translations of units between energy domains. VHDL-AMS encourages concentration on system rather than component considerations, encapsulates low-level information, encourages hierarchical, evolutionary design and reuse, and provides component designers with concrete specifications for their work. It is compatible with the use of component libraries which are already being developed. In addition, it provides a comfortable path into the MEMS design area for electrical and computer engineers. It also encourages the development of

MEMS tools which interface will with current hardware / software design tools. It encourages decoupling of system design from low-level physical considerations, while providing support for simulations which take physical behavior into account. It should be possible to provide VHDL-AMS interfaces to powerful MEMS simulation systems[Senturia98] already under development. VHDL-AMS supports simulation of dynamic system behavior and can model both continuous and discrete events, thereby providing support for the simulation of complex physical systems.

3.4. Possible drawbacks

Some limitations of the approach we have described here include the effort required to interface VHDL-AMS to existing simulators, the present lack of graphical interfaces for VHDL-AMS modeling and simulation, and the lack of symbolic computation. Some flaws in VHDL itself have been pointed out[Ghosh,1999], which may necessitate some redesign both of VHDL and of VHDL-AMS. But we believe that the advantages listed above are more than sufficient to justify our methodology.

4. MATHEMATICA

Mathematica is a useful tool for those who do quantitative analysis, symbolic calculations and manipulations, as well as for those who want to visualize functions or data [Wolfram, 1991]. With it one can calculate, model, prototype, and analyze results.

Mathematica is an interpreted language. This means that it reads an expression, evaluates the result, and then prints it out. Mathematica is programmable. One is able to create functions on one's own. Mathematica has built into the language many of the primitives and constructs found in C, FORTRAN, and Pascal. In addition to procedural programming, Mathematica supports rule-based programming using pattern matching.

Mathematica performs three basic types of computation: numerical, symbolic, and graphical. It works with numbers of arbitrary magnitude and precision, as well as with polynomials, power series expansions, matrices, and graphs. Mathematica provides standard symbolic operations of algebra and calculus, including integration and differentiation.

Version 3.01 was the version used for this project. This package was used to create a template file which contained the equations for the model of the cantilever beam. This file accepts the parameters of the beam that are extracted from the layout file.

4.1. MechanicsExplorers: Add on package for Mathematica

This is a program for the bending of beams. This package uses the Euler-Bernoulli theory for small deflections of thin elastic straight beams. The basic function of the package is *SolveBeam*. It calculates the shear force S_y, the bending moment M_z, and the deflection d and slope s for beams with given loads (discrete forces, distributed forces, and discrete moments), supports (fixed or simple) and hinges. The bending stiffness EI_z can be described by arbitrary functions. E is Young's modulus, and I_z is the main moment of inertial about the z-axis.

The solution is calculated by integrating the following well known differential relations (f_z is the distributed force in z-direction):

$$\delta S_y(x) == -f_z(x)$$

$$\frac{\delta M_z(x)}{\delta x} == -S_y(x)$$

$$\frac{\delta^2 d(x)}{\delta x^2} == \frac{M(x)}{EI_z}$$

Using this package a template was generated which allows us to enter the parameters that are extracted for the cantilever beam.

5. Sample Result

Note: complete results are available in Dennis Gibson, Integrating Behavioral modeling & simulation for MEMS components into CAD for VLSI, M.S. thesis, University of Cincinnati, August 2002.

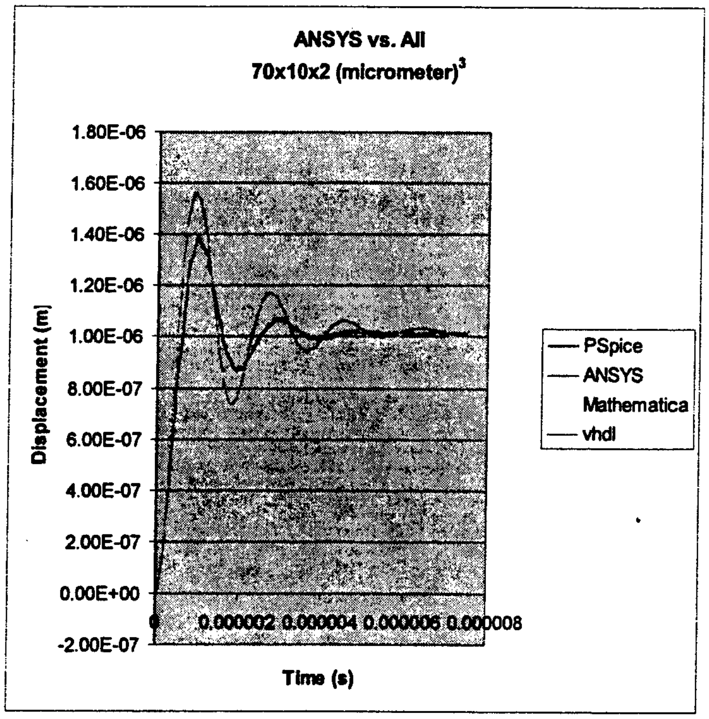


Figure 3.

References

[Gibson, 1999]. D. Gibson, A. Hare, F. Beyette, Jr., and C. Purdy, Design automation of MEMS systems using behavioral modeling, Proc. Ninth Great Lakes Symposium on VLSI, Ann Arbor, Mich. (ed. R.J. Lomax and P. Mazumder), March 1999, 266-269.

[Gibson, 2002].* Dennis Gibson, Integrating Behavioral modeling & simulation for MEMS components into CAD for VLSI, M.S. thesis, University of Cincinnati, August 2002.

[Ansys,1991] Ansys **Primer for Stress Analysis for revision 4.4**, Swanson Analysis Systems, Inc., P.O. Box 65, Johnson Rd., Houston, PA, 15342-0065, copyright 1991.

[Ashenden, 1990] P. Ashenden, "*The VHDL Cookbook, First Edition*", copyright 1990, Dept. Computer Science, University of Adelaide, South Australia.

[Banks, 1996] `http://www.ee.surrey.ac.uk/Personal/D.Banks/ueng.html`, "Introduction to Microengineering", copyright 1996.

[Frey, 1998] P. Frey and R. Radhakrishnan and H Carter and P. Wilsey, *"Optimistic Synchronization of Mixed-Mode Simulators"*, Proceedings of the 1st Merged International Parallel

Processing Symposium and Symposium on Parallel and Distributed Processing, pp. 694-700, ISBN 0-8186-8403-8, March 30-April 3, IEEE Computer Society, Los Alamitos, 1998.

[Standards,1997] Design Automation Standards Committee, IEEE Computer Society, **IEEE Standard VHDL Language Reference Manual (Integrated with VHDL-AMS Changes)**, Std. 1076.1, IEEE, 1997.

[Tuinenga,1992] Tuinenga, Paul W., **SPICE, A Guide to Circuit Simulation and Analysis Using PSPICE**, Prentice Hall, Inc.,copyright 1992, pp. 1-6, 115-132, 147-152.

[VHDL] http://vhdl.org/vi/analog/wwwpages/tutorial/ppframe.htm, *"Analog and Mixed-Signal Extensions to VHDL Through Examples"*.

[Wolfram, 1991] Wolfram, Stephen, **Mathematica: A System for Doing Mathematics by Computer**, Addison-Wesley Publishing Co., Inc., copyright 1991.

TASK 9

An Optimized Native Code Parser & Simulator

Principal Investigator: Prof. Santosh Pande

Institution: University of Cincinnati, Cincinnati, Ohio (now on faculty at Georgia Institute of Technology)

Period of Performance: March 24, 1998 to November 30, 1998

Task: The contractor shall develop, evaluate, and distribute a translator which, given the intermediate representation of SEAMS (Simulator Environment for AMS), produces equivalent efficient GNU triples intermediate form. The GNU triples can then be directly converted into the native code improving the speed and efficiency (in terms of the execution speed of the resulting code) of the translation process.

This task was accomplished by Prof. Joseph Boyd following transfer of Prof. Pande to Georgia Tech during the period of performance. The enclosed report is from Prof. Boyd.

Revised 9/21/98

Freestanding Micromachined Multimode Silicon Optical Waveguides at $\lambda = 1.3\ \mu m$ for MEMS Technology*

Kevin E. Burcham and Joseph T. Boyd
Department of Electrical and Computer Engineering and Computer Science
University of Cincinnati, Cincinnati, OH 45221-0030

Abstract

Freestanding multimode optical channel waveguides formed by micromachining silicon have been demonstrated. Fabrication utilizes standard microelectromechanical systems (MEMS) technology. Losses in the range 0.57 – 0.80 dB/cm have been measured for channel waveguides with an air–silicon–air structure, while losses in the range 1.12 – 1.52 dB/cm have been measured for channel waveguides with a SiO_2–silicon–SiO_2 structure. Freestanding channel waveguides, along with optical fibers and other MEMS structures, can readily be mounted on a silicon MEMS platform to provide optimal alignment for maximizing optical coupling, and are thus expected to be useful in devices which involve light and MEMS structures.

Low-loss freestanding multimode optical channel waveguides fabricated by micromachining silicon utilizing standard microelectromechanical systems (MEMS) technology are demonstrated. Silicon waveguides, having been first demonstrated in epitaxial layers using doping variations to achieve index changes [1], can be operated at $\lambda = 1.3\ \mu m$. Silicon on insulator (SOI) optical waveguide structures [2-4] provide an alternative which eliminates the presence of free carreir absorption. Silicon cantilevers formed by surface micromachining were formed on such SOI structures [5] as were devices for wavelength division multiplexing [6,7]. Large core SOI waveguides have also been demonstrated [8]. The freestanding waveguides reported here are large core, formed by bulk micromachining, and have low propagation losses in comparison to previous silicon waveguides. Our freestanding silicon channel waveguides as well as optical fibers and other MEMS structures can readily be mounted on a silicon MEMS platform to provide optimal alignment for maximizing optical coupling. These waveguides are thus expected to be useful in micro-optical-electro-mechanical systems (MOEMS) and could also be used in micromachined micro-optical systems [9].

The freestanding silicon multimode optical channel waveguide structure being reported here is illustrated in Figure 1. The ridge structures are similar to that commonly used in forming channels except that these multimode structures are much larger and have sloped ridge sidewalls. These sloped sidewalls are formed by bulk micromachining so that the resulting sidewall surfaces, corresponding to <111> planes, are relatively smooth for etched surfaces. As an example of an application of these structures, Figure 2 illustrates a cantilever device in which such waveguides, along with input and output fibers, are supported on a common MEMS platform designed to provide optimal alignment. In this particular cantilever device the light output depends on, and is thus a measure of, the position of the free end of the cantilever waveguide. This configuration could thus be used to monitor acceleration or displacement.

* Research supported by NASA and the Air Force Research Laboratory (AFRL/IFTA).

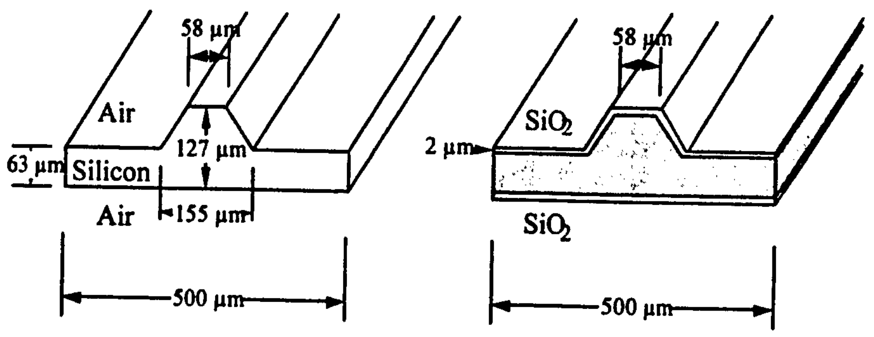


Figure 1. Channel waveguide configuration and typical dimensions for both the air-silicon-air and the SiO_2-silicon-SiO_2 configurations

We determined the form of the electric field distribution associated with light propagating in the freestanding channel waveguide structure shown in Figure 1 using BeamProp [10]. The input field profile was obtained by measuring the output from a multimode fiber (100/140 µm core/cladding) which had been coupled with light from a 1.3 µm laser source. Figure 3 illustrates the output profile from the end of the silicon channel waveguide after propagation over a length of 8000 µm. Note that the resulting light distribution is laterally confined within the trapezoidal ridge as was expected.

To fabricate the channel waveguide structures shown in Figure 1., a 2 µm thick thermal oxide masking layer was grown on five, <100>-oriented silicon wafers, ≈ 127 µm in thickness. These wafers were then patterned with channel waveguide ridges using standard photolithography techniques. Typical dimensions for the fabricated channel waveguides are shown in Figure 1. Etching was done in a KOH solution at a temperature of 70°C, proceeding until half of the initial wafer thickness had been etched

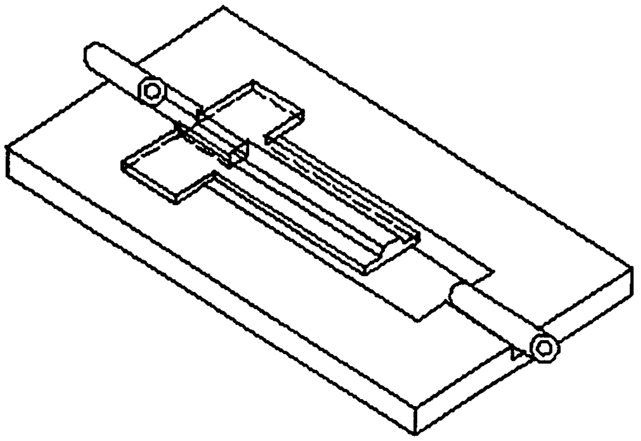


Figure 2. Example of a MEMS platform supporting fibers and freestanding waveguide.

away. The wafers were then given a standard RCA base clean, the masking oxide layer was removed, and the wafers were reoxidized. The oxide was then removed from three of the wafers and left on the

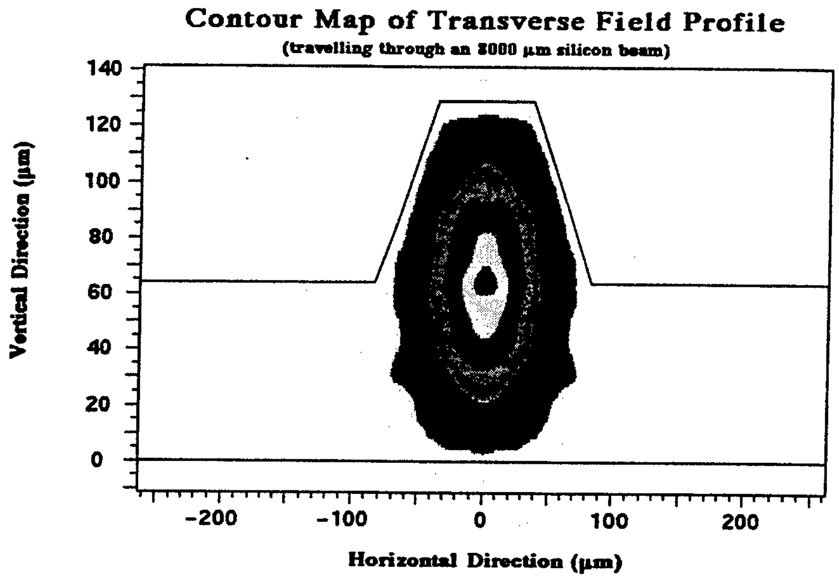


Figure 3. Calculated field profile for the silicon channel waveguide.

remaining two. The wafers were then cleaved perpendicular to the channels to obtain smooth <110> plane surfaces for input coupling. There were sixteen channel waveguides on each sample wafer.

Optical waveguide propagation loss measurements were performed on these waveguides by the out-scattering technique [11]. This technique utilizes a fiber probe to measure light scattered normal to the surface of the waveguide at a number of points along the propagation path of the light. Accuracy in determining propagation loss is enhanced by maintaining a constant distance from the fiber tip to the waveguide surface for each measurement point. In our experiment this is accomplished by imaging the top channel surface using a 8× (0.15 N.A.) long working distance objective, a 3 mm diameter fiber probe with 1.25 mm diameter aperture built into a microscope eyepiece, and a PbS tube infrared camera to observe the waveguide streak through the eyepiece. Imaging is performed by bringing into focus both the waveguide streak and the marker on the fiber probe face plane. The effective diameter of the input aperture of the fiber integrated the outscattered light intensity from an ≈ 145 µm diameter region of waveguide surface at every measurement point. We expect the intensity of the light scattered normal to the waveguide at a given point to be proportional to the intensity of light in the waveguide at that point. Since the intensity of light propagating in the waveguide is expected to decay exponentially on the average, by fitting the measured light intensity to an exponential, a mean decay constant can be determined. This is done by plotting the measured intensity on a log scale as a function of propagation distance and fitting the data to a straight line. The slope of the resulting line yields the waveguide propagation loss, expressed in units of dB/cm. To mitigate the effect of nonuniform defects, this measurement was performed twice on each waveguide channel; coupling once into each end of the channel with the two values averaged[12] to yield an effective propagation loss for each channel. For each of our samples the maximum difference encountered in loss values measured for the two directions was less than 20%. Figure 3 presents data measured in one direction for one of the air–silicon–air waveguide channels, shown in Figure 1–a. The measured loss averaged for the two propagation directions for four similar waveguides varied in the range 0.57 - 0.80 dB/cm. For SiO_2–silicon–SiO_2 channel waveguides, yield values of loss averaged for the two propagation directions for four similar waveguides which varied in the range 1.12 – 1.52 dB/cm.

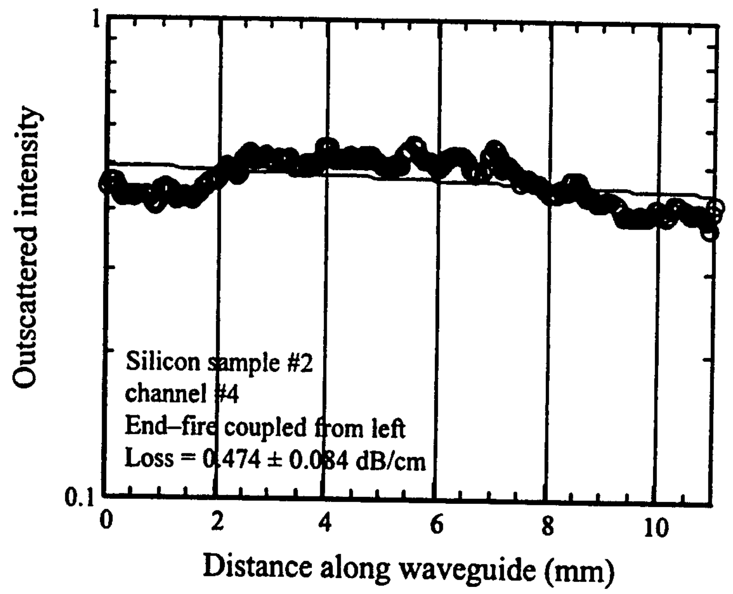


Figure 4. Out-scattering loss measurement data for an air-silicon-air channel waveguide.

In summary, we have demonstrated micromachined multimode freestanding silicon optical channel waveguides having low propagation losses for silicon optical waveguides. Because these channel waveguides, along with optical fibers and other MEMS structures, can readily be mounted on a silicon MEMS platform to provide optimal alignment for maximizing optical coupling, we expect that these waveguides will be useful in devices which involve MOEMS structures.

References

[1] R. A. Soref and J. P. Lorenzo, "Single-Crystal Silicon: A New Material for 1.3 and 1.6 μm Integrated-Optical Components," Electronics Letters vol. 21, pp. 953-954, (1985).

[2] R. A. Soref, J. Schmidtchen, and K. Peterman, "Large Single-Mode Rib Waveguides in GeSi-Si and Si-on-SiO_2," IEEE Journal of Quantum Electronics vol. 27, pp. 1971-1974 (1991).

[3] B. Schuppert, J. Schmidtchen, A. Splett, U. Fischer, T. Zinke, R. Moosburger, and K. Petermann, "Integrated Optics in Silicon and SiGe-Heterostructures," Journal of Lightwave Technology vol. 14, pp. 2311-2323 (1996).

[4] A. Cutolo, M. Iodice, P. Spirito, and L. Zeni, "Silicon Electro-Optic Modulator Based on a Three Terminal Device Integrated in a Low-Loss Single-Mode SOI Waveguide," Journal of Lightwave Technology vol. 15, pp. 505-518 (1997).

[5] T. T. H. Eng, S. C. Kan, and G.K. L. Wong, "Surface-Micromachined Epitaxial Silicon Cantilevers as Movable Optical Waveguides on Silicon-on-Insulator Substrates," Sensors and Actuators A vol. 49, pp. 109-113 (1995).

[6] S. Yegnanarayanan, P. D. Trinh, F. Coppinger, and B. Jalali, "Compact Silicon-Based Integrated Optic Time Delays," IEEE Photonics Technology Letters vol. 9, pp. 634-635 (1997).

[7] B. E. Little, J. S. Foresi, G. Steinmeyer, E. R. Thoen, S. T. Chu, H. A. Haus, E. P. Ippen, L. C. Kimerling, and W. Greene, "Ultra-Compact Si-SiO_2 Microring Resonator Optical Channel Dropping Filters," IEEE Photonics Technology Letters vol. 10, pp. 549-551 (1998).

[8] T. T. H. Eng, J. Y. L. Ho, P. W. L. Chan, S. C. Kan, and G.K. L. Wong, "Large Core (~60 μm) SOI Multimode Waveguides for Optical Interconnects," IEEE Photonics Technology Letters vol. 8, pp. 1196-1198 (1996).

[9] L. Y. Lin, S. S. Lee, K. S. J. Pister, and M. C. Wu, "Micro-Machined Three-Dimensional Micro-Optics for Integrated Free-Space Optical System," IEEE Photonics Technology Letters vol. 6, pp. 1445-1447 (1994).

[10] BeamPROPTM, Rsoft, Inc., 13 Lancaster Avenue, NY 10548, (914-734-2665).

[11] M. Kumar, J.T. Boyd, H.E. Jackson, J.M. Zavada, H.A. Jenkinson, R.G. Wilson, B. Theys, and J. Chevallier, "Channel Optical Waveguides Formed by Deuterium Passivation in GaAs and InP," Journal of Applied Physics vol. 82, pp. 3205-3213 (1997).

[12] Vijaya Subramaniam, Gregory N. De Brabander, David H. Naghski, and Joseph T. Boyd, "Measurement of Mode Field Profiles and Bending and Transition Losses in Curved Optical Channel Waveguides," Journal of Lightwave Technology, vol. 15, pp. 990-997, (1997).

TASK 10

3D Microwave Monolithic Integrated Circuit (3D MMIC) Design

Principal Investigator:	Prof. Altan Ferendeci
Institution:	University of Cincinnati, Cincinnati, Ohio
Period of Performance:	March 24, 1998 to May 31 1999
Task:	The contractor shall develop vertically interconnected 3D MMIC technology for a bilateral UC/Korean effort to be applicable to a broad range of critical defense as well as commercial applications. Particularly, the emphasis will be on modeling and design.

X

3D Multilayer Microwave Integrated Circuit (3DIC)

Quarterly Report (Covering Jan – Mar. 00)

During the last three months, work continued on three areas:

a) Vertical posts and isolation trenches.
b) Cavity backed slot antennas
c) Intermediate level RF MEMS switches.

VERTICAL POSTS AND ISOLATION TRENCHES

For the vertical interconnects/vias, there are two major concerns which effect the performance of interconnected layers: (1) low insertion and return losses, and (2) introduction of negligible field and electrical interference with the rest of the system elements. The latter is more important in communication applications.

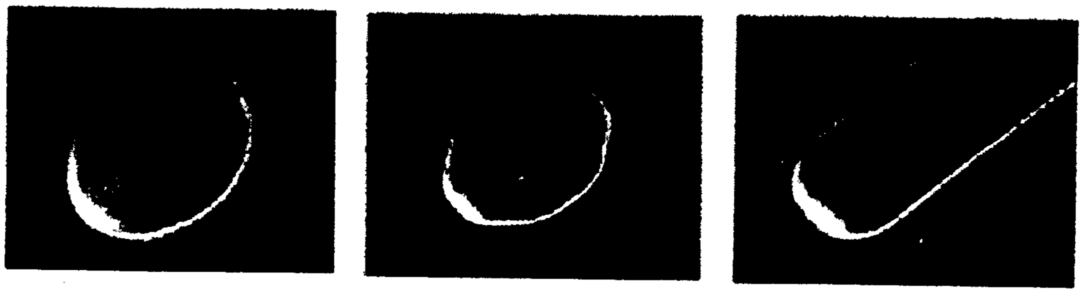
Stripline transmission lines were used at different levels. To simulate the performance of vertical interconnects in the frequency domain, Ansoft HFSS simulation software was used. Surface current distributions on vias and ground planes were obtained. Via and clearance-hole shapes were also compared for their insertion and return losses.

The results of simulations of vertical posts, the resulting insertion loss and a simple equivalent circuit for the posts were presented in our previous report. During the present report interval, efforts were concentrated on metal deposition and surface planarization techniques

. A special mask with various features was prepared. Cr/Au/Cr was deposited over a Silicon substrate. Polyimide of 10 μm thick was spun coated over the metal layer. A second metal layer of Cr/Au/Cr was deposited as a mask layer. Next photoresist is deposited over the metal. Then the mask was exposed and the photoresist was developed.

Using RIE, the polyimide was etched until the bottom metal layer was exposed. The photoresist and the top metal layer acting as a mask were then chemically removed. Figure 1 show various geometrical openings after the RIE.

Gold was then electrodeposited to fill the openings. The final shape of the deposited metal was like a mushroom shape as shown in Fig.2.



(a) (b) (c)

Figure 1.RIE etched polyimide. Single layer (10 μm). a)Circular, b)Octagonal and c) trench openings.

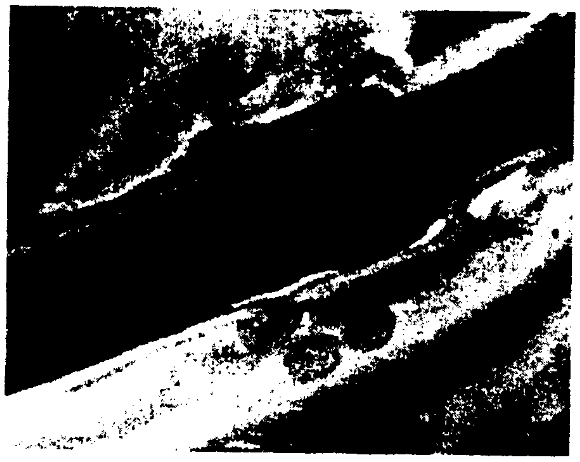


Figure 2. Mushroom shaped electrodeposited Gold on the trench of Fig1c.

The sample was sent to Lapmaster International for lapping and polishing. The returned sample showed a parallelity of 5 μm. SEM pictures showed the removal of the mushroom shapes. The flatness and finish are being investigated. The results are highly promising in the sense that mechanical lapping and polishing can be controlled within the design dimensions. Planarization problems associated with polyimide/metal depositions will be eliminated by subsequent lapping and polishing whenever there is need for them.

PLANAR ANTENNAS

Antenna Configurations

Two antenna configurations were initially considered.

a) patch antenna
b) cavity backed slot antenna

The maximum polyimide thickness that can be spun coated over the ground plane below the antenna element was limited to 70-100 μm. This required deposition of 10 multiple layers of polyimide. Unless a polyimide which can be deposited to larger thickness in a single process is available, the maximum dielectric layer is limited to this thickness.

With these restrictions,, a rectangular patch antenna was designed to operate at 10 GHz. As expected, the simulated results predicted a bandwidth and efficiency of less than 1% for each of the parameters.

As a second alternative a cavity backed slot antenna as shown in Figure 3 was considered. A cavity is created between the top and bottom ground planes by building four vertical metal walls in the dielectric layers. Since the whole cavity is very thin, the depth of the cavity (44 μm) is much less than one half of the guide wavelength (λ_g) at 13 GHz. Only TM_{mn0} modes can be excited in this case. The cavity is essentially a shorted rectangular waveguide. With the cavity's depth of 44 μm, the closest higher order mode is the TM_{120} that resonates at 13.27 GHz.

FIGURE NOT AVAILABLE AT PUBLICATION TIME.

Figure 3. Cavity backed slot antenna.

The slot antenna was fed by a stripline, which lies horizontally in the middle of the cavity; 22 μm away from both the top and bottom ground planes. The width of the stripline was so determined that the characteristic impedance is 50 ohm. Also the stripline was short-circuited at one of the metal side walls. No matching elements were used. The length of the slot was initially chosen close to one-half of a free-space wavelength, or 11mm. The impedance of the slot was then affected by the width of the slot and feed point of the stripline. By using Ansoft HFSS, the slot antenna was simulated. For the slot with size of Ls = 11 mm, Ws = 1 mm, Dx = 4.613 mm ($0.2\lambda_0$) and Dy = 4.125 mm was closely matched to the 50-ohm stripline. Still the bandwidth of the antenna was limited to 3%.

To increase the bandwidth of the antenna, a spiral antenna is being investigated as a third alternative . The spiral could be either a metal strip or the complementary slot type spiral configuration. In each case, the feeding of the antenna will be through a specially designed balun (proposed by AMCOM) which will provide the 180^0 phase difference between the two feed points.

The strip type spiral is being simulated by Ansoft HFFS. In these simulations, the bandwidth and directivity of the spiral antennas will be determined. The thickness of the dielectric layer will still be 100 μm and these spirals will be ground plane backed configurations. Eventually the complementary slot type will also be simulated and an optimum structure will be determined. The antenna will then be processed and measured for its radiation characteristics.

CPWG to Stripline Transitions

In order to access the test port for antenna characterization, a simple low-loss transition from 50-ohm CPW with ground plane to microstrip, and then to a 50-ohm stripline was used, extending the work of Houdard et al. at W-band frequencies. The transition simulated by Ansoft HFSS and LIBRA resulted in less than 0.02 dB insertion loss. The stripline was fed by a CPWG- stripline transition through a small opening in one of the side walls.

Radiation Patterns

The radiated fields and directivity of the cavity backed slot antenna were simulated using Ansoft HFSS. Results of the simulated directivity are shown in Figure 4.

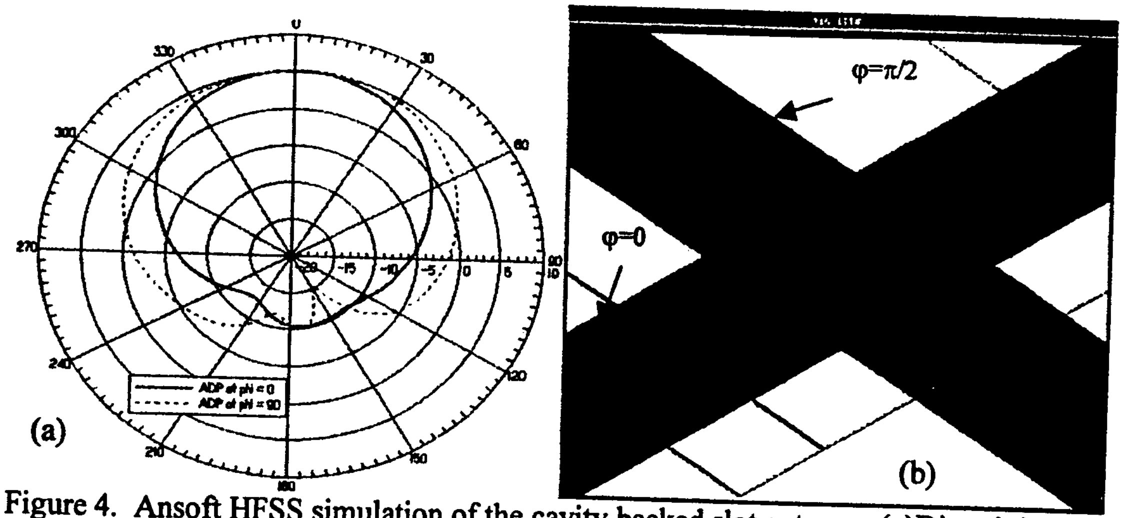


Figure 4. Ansoft HFSS simulation of the cavity backed slot antenna. (a)Directivity in the plane of φ = 0 (-) and φ = π/2 (- -) (b) Electric field magnitudes near the radiating slot

A unidirectional radiation pattern was observed. The electric field magnitude plotted near the slot is also shown in Figure 4.b. E and H plane radiation patterns will be measured..

RF MEMS SWITCHES

A novel RF MEMS switch is being investigated as a means of switching element for the phase shifter that will be placed behind each antenna. The novelty of these switch is that they will be placed in an intermediate layer and will be covered over by the next upper layer of the 3DIC.

Initial designs were made and some sample switches were processed and tested. Some of the problems associated with the processing of the swathes are being investigated further.

In the mean time, the work is initiated on conformal phased array antennas that will utilize the 3DIC modules that are being developed here. Initially, a 2x2 array will be investigated. This will then be extended to a circular array where work will concentrate on scanning multiple beams at the same time as a rotating beam pattern. As the theory and further progress is made on the conformal array concepts, these will be reported in later reports.

FACILITIES:

The work in renovating the Laboratory is being continued. The units that are operational are the sputtering unit, RIE, thermal evaporator, SEM and the anachoic chamber. A bid was received for converting the lab into a clean room. Before final action is taken on this matter, we are waiting the installation of the ductwork and the exhaust fan for the fume hood. We are told that this will take place within the next two weeks.

PUBLICATIONS

1. Altan M. Ferendeci, Bosui Liu, Ho Huang, Missoon Mah, and Lee Liou, "3D Multilayer Monolithic Microwave (M3) passive transmitter module", *IEEE MTT-S Int. Microwave Symp. Dig.*, vol. 2, pp.445-447, June 1999.

2. "Interlayer MEMS RF Switch for 3D MMICs," Yu Albert Wang, Qinghua Kang, Bosui Liu, Altan M. Ferendeci, and Misoon Mah, International MTT-S Microwave Conference 00, Boston, MA (June 12-15, 2000)

3. Vertically Interconnected 3D Module for Conformal Phased Array Antennas Bosui Liu, Yu Albert Wang, George Kang, and Altan M. Ferendeci and Misoon Mah, IEEE International Conference on Phased Array Systems and Technology, Dana Point CA, (May 21-25, 2000)

4. "Vertically Interconnected Multilayer Microwave Circuits", Altan M. Ferendeci, Bosui Liu, George Kang and Misoon Mah, 1999 Government Microcircuit Applications Conference (GOMAC 2000), Anaheim, CA (22-24 March 2000).

5. 3D-IC Hybrid Power Amplifier for Vertically Interconnected Multilayer Phased Array Antenna Module.* Altan M. Ferendeci, Peng Xu, and Misoon Mah, GOMAC 2001, San Antonio Texas, (March 5-8, 2001)

6. 3D Broadband Slotted Spiral Antennas with Thin Dielectric Substrates," Bosui Liu* and Altan M. Ferendeci 2002 IEEE AP-S International Symposium and URSI National Radio Science Meeting, San Antonio Texas, June

7. "Broadband Slotted Spiral Antennas With Thin Dielectric Substrates," B. Liu, A. M. Ferendeci, RAWCON 2002, P1.8, Boston, MA (August 10-14, 2002)

8. "RF MEMS Switches for 3D-IC Phased Array Antenna Modules," Y. A. Wang, A. M. Ferendeci, RAWCON 2002, W4.2, Boston, MA (August 10-14, 2002)

*These works were supported by the US Air Force, Air Force Research Laboratory, Contract Number: F33615-96-2-1945

Graduate Students associated with the 3D-IC project
Altan M. Ferendeci, Advisor

1. Yang Tang (M.S.), *Multilayer Circuitry for MicrowaveCommunication*, (Completed Sept. 1998)
2. Yu Albert Wang, Ph.D., *RF MEMS Switches and Phase Shifters for 3D MMIC Phased Array Antenna Systems*, (Completed June 02)
3. Bosui Liu, Ph.D. *Wide-bandwidth Slotted antennas for Vertically Interconnected Monolithic 3D_IC* (Writing his thesis, expected graduation Fall 02)
4. Quinghua (George) Kang,, Ph.D., *Vertical Interconnects for Vertically Interconnected Monolithic 3D_IC* (Writing his thesis, expected graduation Fall 02)
5. Peng Xu, M.S., *HYBRID X-band Power Amplifier Design for 3D-IC Phased Array Module,* (Will defend his thesis on August 20, 02)

TASK 11, 12

Analog and Digital Mixed Signal Processing Language Base, Integrated Circuit Design

Principal Investigator: Profs. Frank Scarpino and Suengug Koh
Institution: University of Dayton, Dayton, Ohio
Period of Performance: June 19, 1998 to March 18, 1999

The final report for this task is unavailable. It will be included within this report when available.

TASK 13

3D Microwave Monolithic Integrated Circuit (3D MMIC) Design

Principal Investigator:	Prof. Altan Ferendeci
Institution:	University of Cincinnati, Cincinnati, Ohio
Period of Performance:	June 19, 1998 to December 18, 1998
Task:	Develop vertically interconnected 3D MMIC technology for a bilateral UC/Korean effort to be applicable to a broad range of critical defense as well as commercial applications. Particularly, the emphasis will be on mumerical analysis using FDTD, modeling and circuit design.

3D Multilayer Microwave Integrated Circuit (3DIC)

Quarterly Report (Covering Oct – Dec. 99)

During the last three months, most of the time was spent in designing, simulation, processing and characterization of various circuit elements. Three areas were addressed:

- Vertical posts and isolation trenches.
- Cavity backed slot antennas
- Intermediate level RF MEMS switches.

I. VERTICAL POSTS AND ISOLATION TRENCHES

I.1. INTRODUCTION

Microwave integrated circuit (MIC) and monolithic microwave integrated circuit (MMIC) packaging technologies are the key design concerns for miniaturizing conventional RF circuits/ devices and for developing newly-emerging microwave applications of compact size. For the multilayer packaging of microwave or millimeter wave circuits, vertical interconnects play an important role in interconnecting circuits of different levels. This allows the planar circuits to be stacked vertically in three dimensions (3D). So the performance of vertical interconnects becomes a big design issue in MMIC packaging. As frequency increases, the propagation characteristics of vertical interconnects or via-holes have an ever stronger effect on the performance of the interconnected electronic circuits. Recently, considerable research efforts have been attracted to address this issue

For the vertical interconnects/vias, There are two major concerns which effect the performance of interconnected layers: (1) low insertion and return losses, and (2) introduction of negligible field and electrical interference with the rest of the system elements. The latter is more important in communication applications.

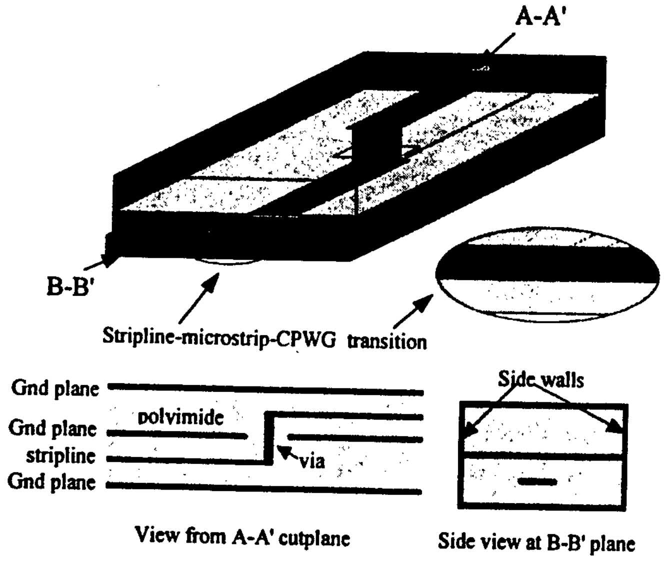
In this paper, an innovative vertical interconnect module is presented. In a vertically interconnected 3D module, vertical posts were used to interconnect circuits at one level with the subsequent upper or lower level. Stripline transmission lines were used at different levels. To simulate the performance of vertical interconnects in the frequency domain, Ansoft HFSS simulation software was used. Surface current distributions on vias and ground planes were obtained. Via and clearance-hole shapes were also compared for their insertion and return losses.

I.2. DESIGN CONSIDERATIONS

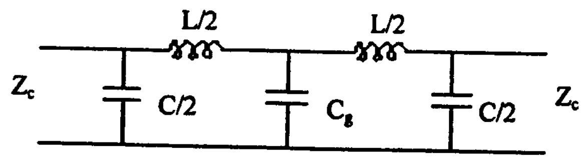
In this work, polyimide (PI1111, DuPont) was chosen as the dielectric material because of its outstanding electrical properties. The dimensions of stripline were calculated first at 10 GHz using HP Libra. Initially losses introduced by the materials were ignored. Metals were treated as perfect electrical conductors (PEC) and metal thin layers as sheets without thickness. This was helpful to establish an equivalent circuit model for the vertical interconnects.

Figure 1 shows a typical structure that was used. In this plot, our attention was focused on the vertical interconnect and trenches, i.e., side walls connecting ground planes along the edges of the dielectric region. The magnified region is the stripline - microstrip - CPWG transition designed for making measurements. To test such a structure, there were inherent difficulties in accessing the striplines at different levels. Special ports had to be designed to access these transmission lines. Special care was taken in the design of the processing steps so that these ports would be accessible for characterization purposes.

The dielectric thickness between the adjacent ground planes was chose 44 μm. The width of center strips was 24 μm and each strip was 1012 μm long (~$\lambda_g/16$, including the via). Thus, the height of the via was 44 μm. Two shapes of via and clearance-hole have been considered: For the vertical posts: a cylindrical (d x H = 24 μm x 44 μm) or a cuboid (L x W x H = 24 μm x 24 μm x 44 μm) via, and for the clearance holes: a circular (d = 50μm) or square (L x W = 50 μm x 50 μm) opening were chosen. The whole simulation size was 960 μm x 2000 μm x 88 μm. For comparison, a via-hole interconnect (Via #5 in Table 1) with all the same structure as others (vs. Via #2) but the confining trenches has also been simulated.



(a)



(b)

Figure 1. (a) *Geometry of the post connecting vertically stripline to stripline transmission lines. Side walls (trenches) are placed along the sides. Stripline-microstrip-CPWG transitions were designed for measurement ports.* (b) *equivalent circuit of a vertical post.*

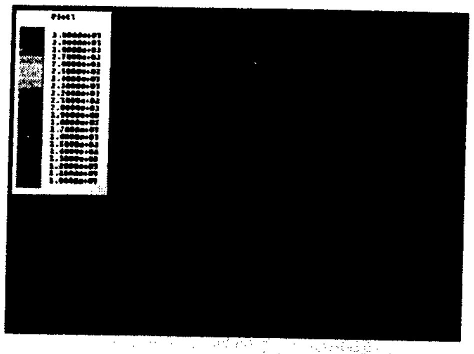
Table 1. Results of different via-holes

Via	Via shape	Clearance hole shape	Insertion loss (dB)
1	Cylinder	Square	0.01480
2	Cylinder	Circle	0.01459
3	Cuboid	Square	0.01728
4	Cuboid	Circle	0.01646
5*	Cylinder	Circle	0.03132

* Without side wall.

I.3. RESULTS

The simulation results of all the interconnects with different structures at frequency of 10 GHz were summarized in Table 1. Via #1 to #4 had the confining trenches and Via #5 did not. To compare the insertion loss, confining trenches also lowered the insertion loss significantly. By taking a closer look at the via shape and clearance hole combinations, a conclusion can be drawn that a cylindrical via and a circular clearance hole both provide a lower insertion loss. The reason may come from that geometric structures of cylindrical and circular shapes which have lower degree of discontinuity than those of cuboid and rectangular shapes. When the mixed combination of via and clearance hole shapes were considered, via shape dominated. The typical surface current distributions on the via surface, and on the center ground plane/strips are shown in Figure 2. Thus the combination of cylindrical via and circular clearance hole gave rise to the lowest insertion loss in this study. The confinement significance of side trenches was far overshadowed the geometry effects (Via #5 vs. others).


(a)

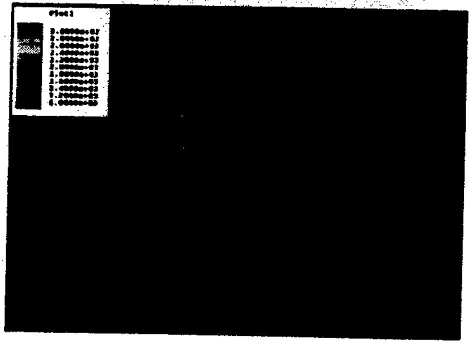

(b)

Figure 2. *Surface current distributions on (a) the via surface only, (b) besides the via, on the strips and separate ground plane.*

Experimental measurements and simulations are being carried out to systematically characterize the vertical interconnect structures. Equivalent circuit elements shown in Figure 1 (b) will be extracted to electrically model the vertical interconnects. Details of the design parameters, experimentally measured results and processing steps will be presented.

II. CAVITY BACKED ANTENNA

II.1 ANTENNA CONFIGURATION

Figure 1 shows the perspective and the top view of the cavity-backed slot antenna. The antenna configuration and design guidelines are summarized below. A design frequency of 13 GHz was chosen.

II.1.1. CAVITY

As shown in Figure 1, a cavity is created between the top and bottom ground planes by building four vertical metal walls in the dielectric layers. The field distribution in the slot is dependent on the excitation of higher cavity. Since the whole

FIGURE NOT AVAILABLE AT PUBLICATION TIME.

cavity is very thin, the depth of the cavity (44 μm) is much less than one half of the guide wavelength (λ_g) at 13 GHz. Only TM_{mn0} mode can be excited in this case. The cavity is essentially a shorted rectangular waveguide. The cut-off frequency of the waveguide was chosen to be 8.3 GHz for TM_{11}, which gives a dimension of 15 mm x 14.5 mm. With the cavity's depth of 44 μm, the closest higher order mode is then TM_{120} that resonates at 13.27 GHz.

II.1.2 STRIPLINE OFFSET-FED SLOT

The slot antenna was fed by a stripline, which lies horizontally in the middle of the cavity; 22 μm away from both the top and bottom ground planes. The width of the stripline is so determined that the characteristic impedance is 50 ohm. Also the stripline was short-circuited at one of the metal side walls. In the initial design, no matching elements were used. The length of the slot was initially chosen close to one-half of a free-space wavelength, or 11mm. The impedance of the slot was then affected by the width of the slot and feed point of the stripline. By using Ansoft HFSS, the slot antenna was simulated as a function of width (Ws = 0.4 Ls, 0.2 Ls, 0.1 Ls, and 0.05 Ls) and feed point (Dx = 0, $0.125\lambda_0$, $0.1875\lambda_0$, $0.21\lambda_0$, $0.225\lambda_0$) of the slot. And it was found that the slot with size of Ls = 11 mm, Ws = 1 mm, Dx = 4.613 mm ($0.2\lambda_0$) and Dy = 4.125 mm was closely matched to the 50-ohm stripline.

II.1.3) CPWG TO STRIPLINE TRANSITION

In order to access to a test port for antenna characterization, a simple low-loss transition from 50-ohm CPW with ground plane to microstrip, and then to a 50-ohm stripline was used, extending the work of Houdard et al. at W-band frequencies. The transition simulated by Ansoft HFSS and LIBRA resulted in less than 0.02 dB insertion loss. The stripline was fed by a CPWG- stripline transition through a small opening in one of the side walls.

II.2. EXPERIMENTAL RESULTS

A cavity-backed slot antenna was designed and fabricated to operate at 13 GHz. Polyimide (PI1111, DuPont) was chosen as the dielectric material with a dielectric constant of 3.0. Referring to Figure 1, the cavity size was L_x=15 mm L_y =14.5 mm and L_z = 44 µm. It was surrounded by metal side walls and filled with 2 layers of hard-cured polyimide. The slot size was Ls=11 mm and Ws=1 mm and was located Dy = 4.125 mm from the edge of the cavity. A 50-ohm stripline feeds the slots at Dx = 4.613 mm and Dy = 4.125 mm. The width of the stripline was 26 µm. The width of the CPWG center strip was 51.8 µm and the gap was 25 µm. All conductors were sputtered gold with a thickness of 2 µm, corresponding to 3 skin depths at 13 GHz.

II.2.1. PROCESS GUIDELINE

The following processing steps were used:

- Lower ground plane metal was Cr/Au/Cr layers of thickness 0.05 µm, 2 µm and 0.05 µm respectively. They were sputtered on 50 mm Si wafer. Cr served as the adhesion layer.
- As the bottom dielectric layer, 2 layers of polyimide (PI1111, DuPont) was spin-coated, and then hard cured. Total thickness of the polyimide layers was 22 µm.
- Cr/Au layer of 0.05 µm and 2 µm, respectively, was sputtered and then patterned by lithography and wet etching to form feeding stripline and CPWG pad.
- Another 2 layers of polyimide (PI1111, DuPont) was spin-coated over the circuit plane, and then hard cured. Total thickness of the polyimide layers was also 22 µm.
- Trenches in polyimide layers were opened by reactive ion etching (RIE) using $Ar/CF_4/O_2$ mixture as the chemical agent and sputtered gold as a mask. The depth of the trenches was 44 µm so that the lower ground plane was exposed in the trenches. A 2 µm thick gold was then sputtered to form metal walls. The top ground plane was formed at the same time.
- Using the top ground plane as a mask, another RIE is performed to remove polyimide layer above the CPWG pads. The polyimide layer that was etched away by RIE was 22 µm.
- And finally, lithography and wet etching was used to open the slot on the top gold layer.

II.2.2. INPUT IMPEDANCE

The input impedance of the cavity backed slot antenna was measured using an HP 8510 network analyzer. The measured input impedance of the antenna is shown in Figure 2. The magnitude of S11 is – 42.9 dB at 13.35 GHz. The insertion loss is nearly –10 dB in the measured frequency range. It is in agreement with design frequency. Using cavity and slot dimensions mentioned above, simulations using Ansoft HFSS predicted S_{11} of – 8.7 dB at 14.36 GHz with no significant insertion loss. This was expected since perfect conductor approximation was used in the simulations.

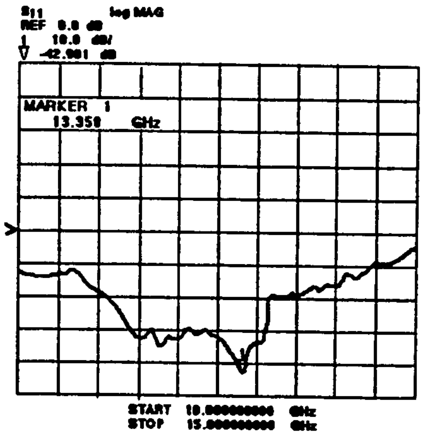


Figure 2. S_{11} of the cavity bcaked antenna

II.2.3. RADIATION PATTERNS

The radiated fields and directivity of the cavity backed slot antenna were simulated using Ansoft HFSS. Results of the simulated directivity are shown in Figure 3.

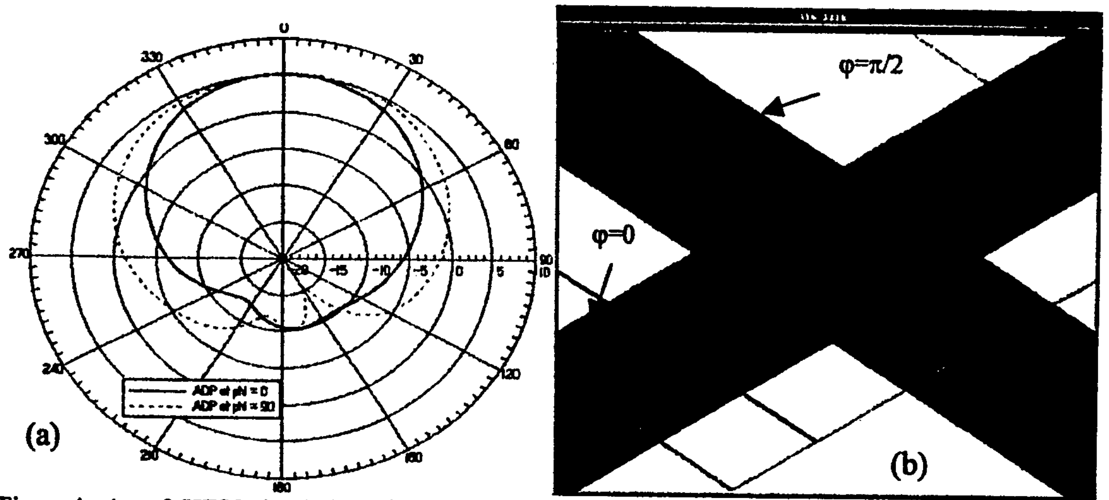


Figure 4. Ansoft HFSS simulation of the cavity backed slot antenna. (a)Directivity in the plane of $\varphi = 0$ (-) and $\varphi = \pi/2$ (- -) (b) Electric field magnitudes near the radiating slot

A unidirectional radiation pattern were observed. The electric field magnitude plotted near the slot were also shown in Figure 3. Due to the finite dimensions of the ground plane, ripples will be expected when measuring the radiation patterns of the fabricated slot antenna [6]. E and H plane radiation patterns are being measured.

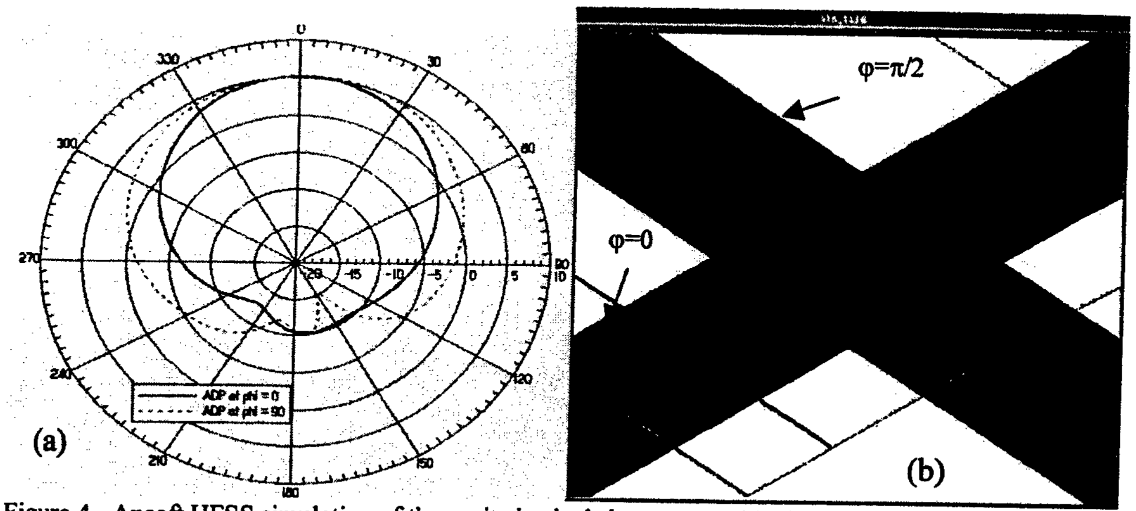


Figure 4. Ansoft HFSS simulation of the cavity backed slot antenna. (a)Directivity in the plane of φ = 0 (-) and φ = π/2 (- -) (b) Electric field magnitudes near the radiating slot

TASK 14

Robust Mixed-Signal Design

Principal Investigator:	Prof. Mohamad Ismail
Institution:	Ohio State University, Columbus, Ohio
Period of Performance:	January 14, 1999 to January 13, 2000
Task:	Establish state-of-the-art research modeling, simulation and design activities with Information Directorate and DAGSI schools in the area of C4ISR Systems and robust mixed-signal design. Develop new leading edge data compression and design integration methodology techniques in the area of C4ISR projects. Integrate new projects into the CERC and II Consortium activities for future DARPA funding. Enhance current Air Force information compression techniques and adaptive multilevel classification in the area of low power smart imagers for C4ISR systems.

TASK 15

Mixed-Signal Adaptive Filtering in Mobile Communications

Principal Investigator: Prof. Raymond Siferd

Institution: Wright State University, Dayton, Ohio

Period of Performance: January 14, 1999 to January 13, 2000

Task: Adaptive equalization is one of the key tasks of a receiver in a mobile communication environment. It mitigates interference originating from multipath propagation. Transversal adaptive equalizers with least-mean-square coefficient adaption have been successfully applied to many communication applications. Adaptive equalizers may be realized in fully digital, fully analog, or mixed signal implementations.

r'

CMOS Analog Implementation of A Discrete-Time 9-Tap FIR Filter with Circular Buffer Architecture

Rong Wang, Ray Siferd, and Robert L. Ewing
Rong.wang@intel.com rsiferd@cs.wright.edu Robert.Ewing@sensors.wpafb.af.mil
Department of Electrical Engineering
Wright State University, Dayton, OH 45435
November 2001

Abstract. This paper presents the design and simulation mixed-signal system. The system is composed of a 9-Tap CMOS Analog Discrete-Time Finite Impulse Response (FIR) Filter. This unique design features a Circular Buffer Architecture which achieves high sampling rate that can be easily expanded to improve speed and extended to higher order filters. Novel area-efficient four quadrant CMOS analog adder and multiplier circuits are employed to respond for high frequency and wide linear range inputs. The Layout for all circuits has been realized using the design tool MAGIC with a 1.2um CMOS process. The performance for each circuit and the whole system are characterized using HSPICE simulation based on the extracted netlist. The 9-tap filter can achieve 5 MHZ sampling rate. The implemented design requires a total chip area of 1690.9um x 2134.2um and ±5 volt power supply.

Key Words. Analog CMOS, analog arithmetic circuits, analog adder, analog inverter, analog multiplier, analog signal processing, discrete-time finite impulse response filter.

1. Introduction

In the information sphere domain, a filter is a system that can be used to modify, reshape, or manipulate the frequency spectrum of a signal according to some prescribed system level requirements. It is widely used in the fields of communication, radar, biomedical system as well as video processing. Depending on the continuous or discrete nature of input, output, and internal operating signals, there are three general types of filters that can be identified; continuous-time, sampled-data, and discrete-time filters. Continuous-time and sampled-data filters are always analog filters. However, discrete-time filters can be analog or digital. Digital filters use binary mathematical operations on the signal to be filtered. The main advantages for the uses of digital filters are high accuracy, ease of design, and noise reduction. However, it needs *A/D* and *D/A* converters since signals are analog in nature and the digital building blocks are usually larger than the analog counterparts. Many applications require high-speed low power equalizers with moderate resolution such as modern magnetic storage channels which usually use partial-response maximum-likelihood detection. For such applications, the analog filter is more suitable because of no requirements for *A/D* and *D/A* converters, the reduced size of some main building blocks, and relatively fast convergence. Discrete time analog filters can be used in adaptive systems, which can learn the signal characteristics and track slow changes to adjust the coefficients. Adaptive filters can be more useful when there is uncertainty about the characteristics of a signal or when these characteristics change. The finite-impulse response (FIR) filter is characterized by a unit-sample response that has a finite duration. One of the advantages of FIR filters is that they can be designed with exact linear phase characteristics and therefore are inherently stable because they have no poles. These filters are the most frequently used in adaptive systems. For example, the application of the least-mean-square (LMS) algorithm to the control of FIR filter gives the approximation of the Wiener solution [1].

2. Principle of Operation

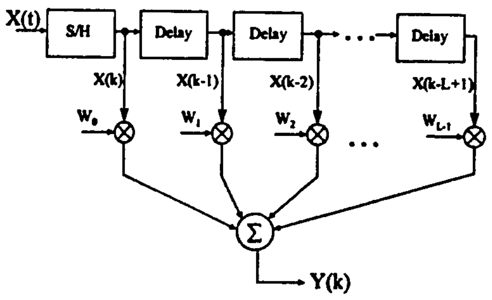
The main goal of this design is to simulate, and analyze a discrete-time fully analog 9-tap finite impulse response (FIR) filter. A circular buffer architecture with an array of sample and hold circuits is used in this design. [2],[3],[9]. Compared to a direct-form or transposed FIR filter, the circular buffer architecture can achieve higher-speed operation since the Sample-and-Hold (S/H) cells can have more than one sample period for acquisition and hold settling. This facilitates a higher sampling rate. Also the architecture does not require the standard delay line structure where errors accumulate as the analog samples passed down the delay line. Another key component of this design is unique CMOS analog circuit for multiplying which consists of analog adding, subtracting, and add inverting circuits[7]. Analog addition, subtraction, and add inversion are typically accomplished with help of operational amplifier based circuits with resistors or their switched capacitor equivalents[4]. The CMOS circuit designs are introduced to implement these functions directly based upon the inherent square law of the MOS transistor drain current when operating in the saturation region. These designs feature differential inputs and require biasing transistors for establishing quiescent conditions for operation in the saturation region. The circuits use single ended voltage inputs, produce single ended voltage outputs, and are self-biasing. This facilitates a wider range of input and output voltages, while keeping the transistors in the saturation region. A four quadrant multiplier is realized by directly coupling the adder, subtractor and add/invert circuits with no buffering. Layout designs for all circuits have been realized using the design tool MAGIC and a 1.2 um CMOS process[5]. The performance for each circuit and the whole filter system are characterized using HSPICE simulation based upon the extracted netlist from the layout.

2.1 FIR Filter

The FIR filter is the most practical and widely used implementation of discrete time filter. The reason for this preference is FIR filters are always stable, it has only adjustable zeros. Even more important, FIR filters are capable of perfectly linear phase (a pure time delay), meaning totally freed from phase distortion. The design of FIR filters involves the selection of a finite sequence that best represents the impulse response of an ideal filter. In general, an FIR filter system is described by the difference equation:

$$Y(k)=\sum_{i=0}^{L-1}\{Wi\cdot X(k-i)\} \qquad (2.1)$$

Suppose *X(k)* denotes the sampled input sequence, and the corresponding output is *Y(k)*. The direct-form realization follows immediately from this nonrecursive difference equation. The length of the FIR filter is selected as *L* to conform with the established notation in the technical literature. The W_i is the weight coefficient for each tap. The structure is illustrated in Fig. 2.1. This structure requires *L-1* delays for the *L-1* previous inputs, and has a complexity of *L* multiplications and *L-1* additions per output points. Since the

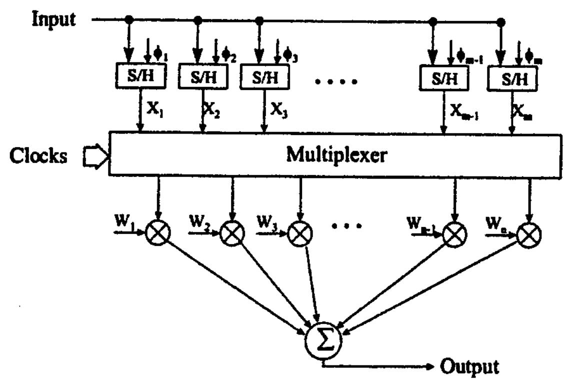


Direct-Form (Tapped Delay Line) Realization

Fig. 2.1 Direct Form (Tapped Delay Line) Filter

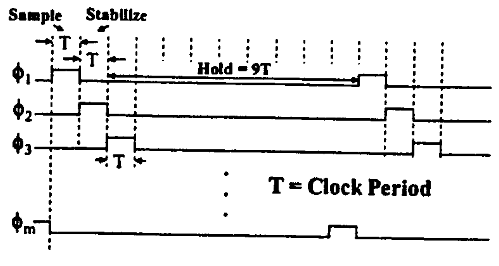
output consists of a weighted linear combination of *L-1* past values of the input and the weighted current value of the input, the structure resembles a tapped delay line or a transversal system. Consequently, the direct-form realization is often called a transversal or tapped-delay-line filter. This structure allows easy expansion (modularity), low complexity and simple timing. However, a main problem associated with this kind of architecture for analog implementation is that the error in the delay line becomes additive, it is accumulated from stage to stage. This can cause unacceptable inaccuracies. Also, the sampling rate is low because it only has one clock period for both acquisition and hold settling. Each tap signal is delayed from previous signal by one cycle.

2.2 Circular Buffer Architecture With the help of circular buffer architecture, we can solve the problems we have in the tapped delay line structure. The circular buffer architecture was first published in 1992[2] and has been used in many applications[3],[8]. Fig.2.2(a) shows the basic structure of the circular buffer. In the circular buffer architecture, an array of parallel *S/H's* are used rather than one *S/H* in the delay line structure to track the input. The number of *S/H's*, *m*, is larger than the number of taps, *n*, so that at anytime, *n* out of *m S/H's* can be used in the calculation of the output samples. By choosing *m* and *n* and using proper *S/H* control clocks, the acquisition and hold settling times of the S/H's can be greater than one sampled period, and high-speed operation is facilitated. The *S/H* control clocks cycle around the array in a circular manner as shown in Fig. 2.2(b) for $m = 11$, $n = 9$, which are used in this design. *T* is the master clock period, so an output sample is produced every *T* seconds. In the time domain, each *S/H* uses one master clock period for tracking the input, another period to settle to a stable held value after the sample-to-hold transition. The stable held value stays unchanged for nine clock periods before the next hold-to- sample transition. At the end of each period, one *S/H* cell changes from sample mode to hold mode, one changes from hold mode to sample mode, and all the others remain unchanged. As a result, nine stable *S/H* output values are available for multiplication at each master clock cycle. Another advantage of this structure, in contrast to a conventional transversal filter, is that all the *S/H* cells track the input directly and the errors do not accumulate as they do in a serial *S/H* delay line. One complication of this architecture is that when a *S/H* is hold mode, its held value X_k becomes X_{k-1} after one clock cycle and must be rerouted to the appropriate multiplier. This rerouting is realized by an m-input, n-output multiplexer, which is an 11 by 9 array of analog switches for this design.



Circular Buffer Architecture Realization

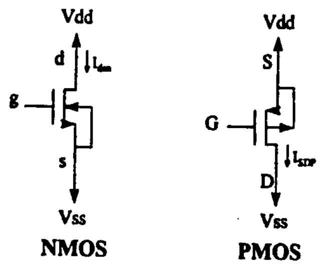
Fig. 2.2(a) FIR Filter with Circular Buffer Architecture



S/H Circular Control Clocks for m=11, n=9

Fig 2.2(b) Clock Timing for FIR Filter

2.3 Inherent Square law of MOS Transistor The multiplication and addition in this design are ealized based on the inherent square law of the MOS transistor drain current when operating



MOS Transistor Diagram

Fig. 2.3 NMOS and PMOS Schematics

In the saturation region. The MOS transistor schematics are shown in Fig. 2.3. The substrates are connected to V_{dd} (PMOS) and device to V_{ss} (NMOS). The drain current for the NMOS transistor operating in the saturation region is

$$I_n = 1/2\beta_n(V_{gs} - V_{tn})^2 \qquad (2.2)$$

where V_{tn} = threshold voltage of n-channel. $\beta_n = \mu_n C_{ox} W/L = K_n W/L$.

From (2.2) we can find an expression for the drain current I_n as a function of the gate voltage, V_g.

$$\begin{aligned} I_n(V_g) &= 1/2\beta_n(V_g - V_{ss} - V_{tn})^2 \\ &= 1/2\beta_n(V_g + C_n)^2 \\ &= 1/2\beta_n(V_g^2 + 2V_g C_n + C_n^2) \qquad (2.3) \end{aligned}$$

where $C_n = -V_{ss} - V_{tn}$.

Similarly for the PMOS transistor, the drain current in the saturation mode is

$$I_p = 1/2\beta_p(V_{SG} - |V_{tp}|)^2 \qquad (2.4)$$

where V_{tp} = threshold voltage of p-channel.

$$\beta_p = \mu_p C_{ox} W/L = K_p W/L.$$

The drain current as a function of the gate voltage for the PMOS transistor is given by

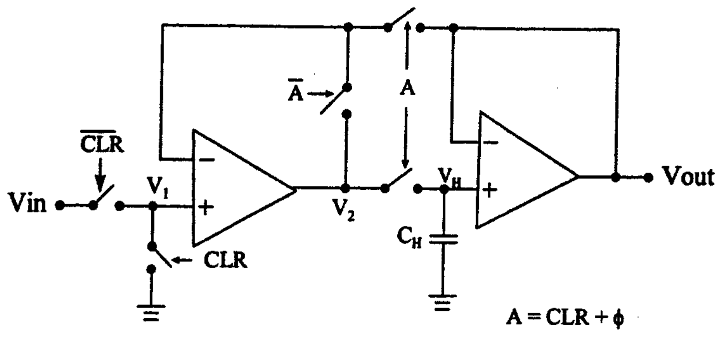
$$\begin{aligned} I_p(V_g) &= 1/2\beta_p(V_{dd} - V_g - |V_{tp}|)^2 \\ &= 1/2\beta_p(C_p - V_g)^2 \\ &= 1/2\beta_p(V_g^2 - 2V_g C_p + C_p^2) \qquad (2.5) \end{aligned}$$

where $C_p = V_{dd} - |V_{tp}|$.

Each multiplication circuit and its subcircuit, such as addition, subtraction, and invert addition circuit are designed to produce a combination of drain currents so that the output voltage is the desired linear function of the input voltages.

3. Building Block Modules [10]

Based on the relevant theories and concepts described in section 2, the system main building blocks are introduced in this section. The functioning of each block module is described, and the performance is also simulated and analyzed. The multiplier module design is one of the most important steps in this filter implementation. It plays a very important role in determining the system area, power dissipation, accuracy, as well as speed. The ideal multiplier should have good linearity, wide range of input and output voltages, low power dissipation, and small size area. In this design, a fully analog multiplier circuit is implemented by coupling the adder, subtractor and add/invert circuits which are based upon the inherent square law of the MOS transistor drain current when operating in the saturation region. After choosing proper adjustment voltages and load resistance, we can get the multiplier circuit suited for this filter design with the expected gain and accuracy. The adder circuit is part of the multiplier circuit. Also, it's used for adding the result of each tap of the 9-tap FIR filter together to get the filter output.



Sample and Hold Circuit

Fig. 3.1 Sample and Hold Using Two Wide Range Differential Amplifiers (WRAmp)

3.1. Sample and Hold Circuit The Sample and Hold circuit is a very critical part of the system. The speed of the whole system is basically decided by the speed and load capacity of the sample and hold cell. In this design, as shown in Fig. 3.1, two voltage followers are designed by using wide range differential amplifiers (WRAmp). Combining several additional CMOS switches controlled by the control signals A and CLR which allows to clear the cell, we can realize the circuit for tracking and holding the input signal.

3.2.1 CMOS Wide Range Differential Amplifier For a standard one stage differential amplifier, the gain is

$$V_{out}/V_{id} = g_{m1}R_{out} = g_{m2}R_{out}. \qquad (3.1)$$

where

g_{m1}, g_{m2} = transconductance of n-channel input transistors,
R_{out} = output resistance of circuit.

The output voltage range is

$$V_2 - V_{Tn} \le V_{out} \le V_{DD} - (2I_{SD4} / \beta_4)^{1/2}. \qquad (3.2)$$

Where

V_{Tn} = threshold voltage of n-channel transistor,
I_{SD4} = quiescent drain current of p-channel,
β_4 = current gain of p-channel transistor = $C_{ox}(W/L)$.

A typical quiescent value for V_2 would be 0 volts, so that the output is limited to values above $-V_{tn}$ volts to keep transistor M_2 in saturation. For the 1.2um process, the value of V_{tn} is about 0.57, so the output is restricted to values greater than -0.57 volts. The schematic of the Wide Range Opamp used in the circuit is shown in Fig. 3.2. It's a one stage differential amplifier combined with several current mirrors.

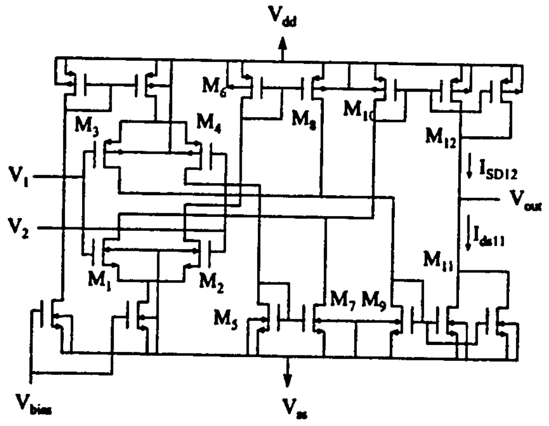


Fig. 3.2 Wide Range Differential Amplifier

The gain of this amplifier is

$$V_{out}/V_{id} = (W/L)_{12}/(W/L)_{10}(g_{m1} + g_{m3})R_{out}. \qquad (3.3)$$

where

g_{m1} = transconductance of n-channel input transistor,
g_{m3} = transconductance of p-channel input transistor.

This is assumes $(W/L)_{12}/(W/L)_{10} = (W/L)_{11}/(W/L)_9$, and $(W/L)_6 = (W/L)_8$, $(W/L)_5 = (W/L)_7$. The output voltage range is

$$V_{SS} - (2I_{DS11}/\beta_{11})^{1/2} \le V_{out} \le V_{DD} - (2I_{SD12}/\beta_{12})^{1/2}. \qquad (3.4)$$

where

V_{DD} = source voltage (5V),
V_{SS} = substrate voltage (-5V).

From equation (3.3), we know that the voltage gain of this WRAmp is proportional to $(g_{m1} + g_{m3})$, which is larger than the one in the standard one stage differential amplifier[10]. Also, the output voltage can swing over a wider range because of the use of current mirrors to develop the single ended output voltage (Eqns 3.2 and 3.4). In this design, V_{bias} was set at -3.9 volts, which results in $I_{dsQ1} = I_{dsQ2} = I_{SDQ3} = I_{SDQ4} = 3.59$ uA, $I_{dsQ11} = I_{SDQ12} = 14.3$ uA. The voltage gain of amplifier with here quiescent condition is 82.5 dB. The output range based on equation (3.4) is -4.31 volts < V < 3.6 volts.

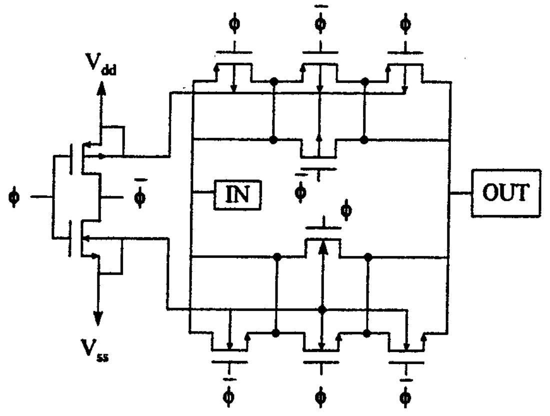


Fig. 3.3 Schematic of Analog Switch Circuit

3.2.3 Switch Circuit The schematic of the switches used in the sample and hold circuit is shown in Fig. 3.3. Also the same configuration switches are employed to build the switch matrix used as the multiplexer in the circular buffer architecture. This design uses dummy transistors connected to the inverted control voltage to apply the opposing feedthrough, which is a flow of charges from the switch's control voltage source to switch terminals through the switch's parasitic capacitance. It can cancel most feedthrough problems which exist in a simple CMOS switch. When the control clock signal ϕ (5V) is employed, the input signal is passed through the switch, equal to the output. Otherwise, if ϕ'(-5V) is applied, the output of the switch is equal to zero.

3.2.4 Final Circuit and Functioning For the sample and hold circuit, initially a very small holding capacitor (C_H) was used to allow high sampling rates. But for the specific load, if the C_H was not big enough, the input signal tracking curve would overshoot, and would not converge at the expected point, resulting in big errors and low accuracy. After many simulations and tests, the final value of C_H = 1.1pF was selected for the 5MHZ clock frequency and around 0.3pF load capacitance.

Signal A is derived from the following:

$$A = CLR + \phi. \tag{3.5}$$

When Clear is high, the circuit is in clear status, and A returns high to allow a zero-volt signal to be loaded into C_H, then transmitted to the output V_o. When Clear is low, the cell samples and holds the input signal. A follows the clock phase signal ϕ. When ϕ is high, the input signal information is sampled into C_H, and passed to V_o, we call it sample phase. The accuracy is ensured due to the feedback path from the output to the input. When ϕ is low, A returns low, the switches controlled by A are open, and the circuit is divided into two parts: the first stage continues to track the input signal, while the output voltage maintains a constant value which is equal to the one being held in C_H in the second stage. This is called the hold phase.

3.2.5 Simulation Results The design is simulated using HSPICE with the circuit load capacitance equal to 0.3pF. The input signal is 200KHZ sine wave with amplitude of 2V. Table 3.1 numerically shows the results and accuracy at several test points. From these, we can see the circuit presents good accuracy (error < 1.00%) and relatively high speed (sampling rate > 5MHZ).

Table 3.1 Sample/Hold Circuit Accuracy Table.

Error = (Output-Input)/4 * 100%(Volts)

Phase	Input/Output	Error %	Input/Output	Error %
1	0.861/0.835	-0.65	-1.99/-1.99	0.00
2	1.47/1.43	**-1.00**	-1.93/-1.91	-0.50
3	1.86/1.84	-0.50	-1.61/-1.59	0.50
9	-0.628/-0.606	0.55	1.94/1.97	0.75
10	-1.28/-1.26	0.50	1.98/1.94	**-1.00**

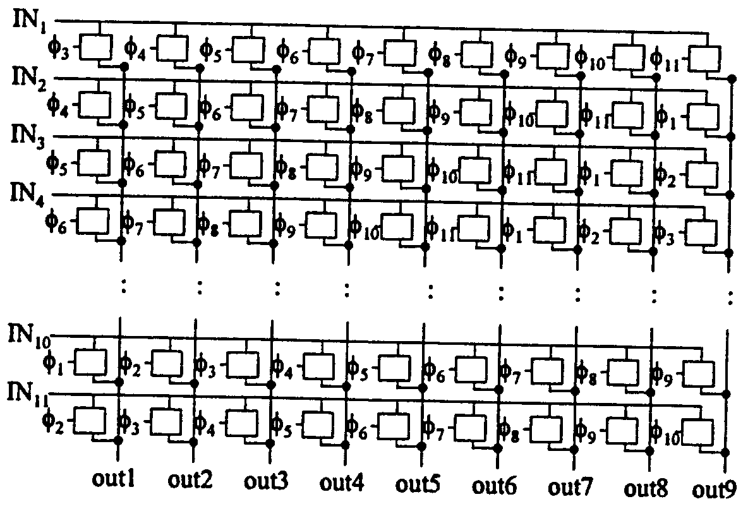


Fig. 3.4 Sketch Map of the Multiplexer

3.3 Multiplexer Module The multiplexer design is showed in Fig. 3.4. It is kind of a switch matrix, which consists of a group of switches mentioned in Section 3.2.3. The inputs of the multiplexer are the 11 phase output samples generated by the *S/H's* array. The outputs are the 9 tap input signal samples which have one clock cycle delay between each tap. The control signals are the 11 clock phase signals generated by the clock generator. From section 2.2 we know, the *S/H's* array consists of 11 *S/H's* to generate 11 phase output samples. For each phase sample, it has one clock period sample time to track the input, one clock period time settle to a stable value after sample-to-hold transition, and nine clock period time to keep the held value unchanged before the next hold-to-sample transition start. The value during these periods is what we want to use to generate the input signal samples for each tap from the 11 phase *S/H's* output samples. For example, the phase-1 sample output is sampled during phase-1, settled down during phase-2, and kept stable during phase-3 to phase-11, so we should use the value of phase-1 output sample during phase-3 to phase-11. In the same way, the phase-2 sample output should be employed during phase-4 to phase-1, and so on. The multiplexer shown in Fig. 3.4 effectively performs this mechanism. For tap1, the output signal generated from the sample values of phase-1 to phase -11 during the time from phase-3 to phase-11 and to phase-2; for tap2, the output generated from phase-2, phase-3... to phase-1 during phase-4 to phase-5 and to phase-3, so do the other taps. Each tap has a one phase delay or one clock cycle delay from the previous one.

3.4 Clock Generation Module The clock generation module is an important device in the system. It synchronizes the whole system and creates control clocks used in *S/H's* and multiplexer module from the input clock signal. The design of this cell is shown in Fig. 3.5. As mentioned before, we need 11 phases of control clocks for the multiplexer, and they are produced by using a flip-flop loop in the clock generator. There are 11 flip-flops in the loop because of the 11 clock period delays needed. The control signals for the 11 *S/H's* are generated by combining the ϕ_i with Clear as shown in Fig. 3.5. The flip-flops are available from standard cells, which are negative edge triggered with set or reset.

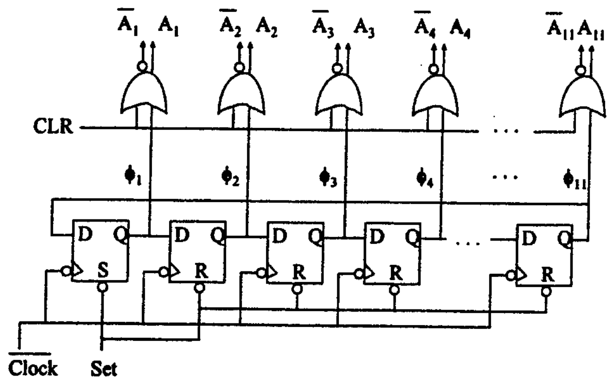


Fig. 3.5 Block Diagram of Clock Generator

4. System Design and Simulation

In section two, the relevant features of the 9-tap circular buffer architecture FIR filter were introduced, and the design of building block modules (analog multiplier module, Sample/Hold module (*S/H*), multiplexer module, and clock generation module) are presented in section three. In this section, the 9-tap circular buffer architecture FIR filter system is designed based on integrating the previous built blocks. In addition, the simulations of the whole filter system as a notch filter are obtained and discussed.

Based on the available modules, the 9-tap FIR filter with circular buffer architecture is constructed, and the block diagram is shown in Fig. 4.1 with a total area of 1.69mm ×2.13mm.

4.1.1 Circular Buffer Architecture Organization To implement the circular buffer architecture mentioned in section 2.2 and Fig. 2.2a, 11 phase clocks created by the clock generation module are used to control an array of *S/H's* to track the input signal successively. By choosing the proper control clocks phase, the multiplexer circuit can generate the required values for the nine taps output which are fed into the multiplier circuit later. In other words, this is a process to put the relative separate modules together to make an effective system.

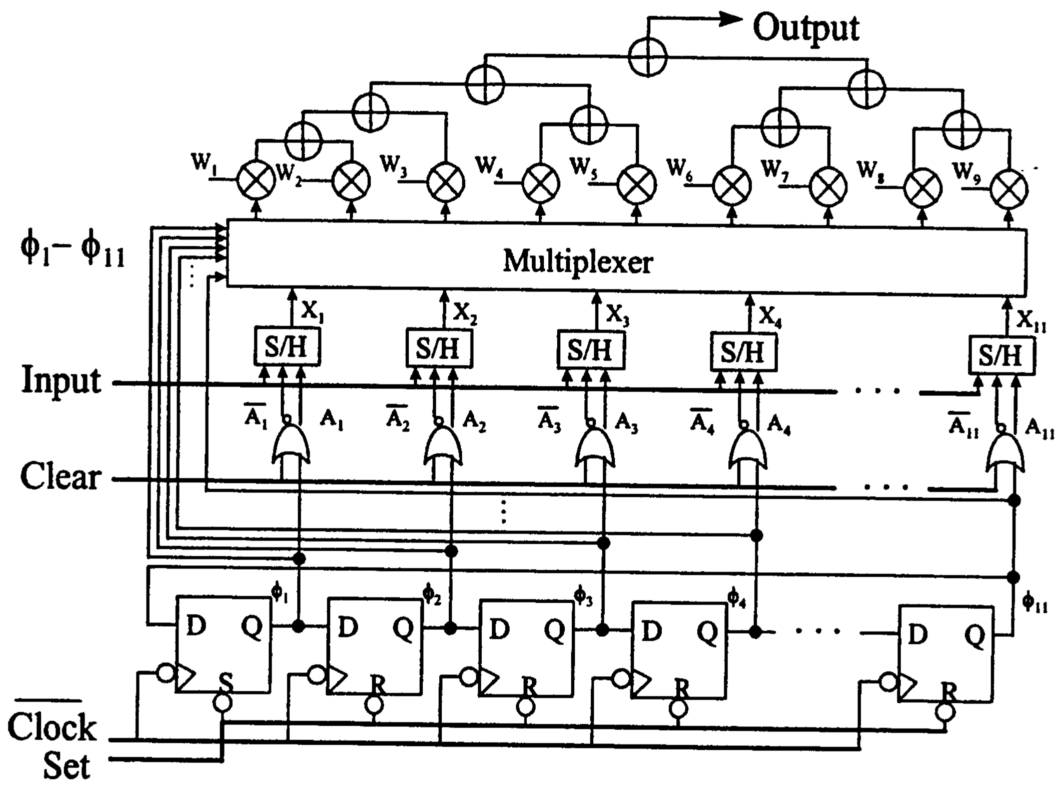

Output
W_1
W_2
W_3
W_4
W_5
W_6
W_7
W_8
W_9
$\phi_1 - \phi_{11}$
Multiplexer
X_1
X_2
X_3
X_4
X_{11}
S/H
Input
$\bar{A}_1$ A_1 $\bar{A}_2$ A_2 $\bar{A}_3$ A_3 $\bar{A}_4$ A_4 $\bar{A}_{11}$ A_{11}
Clear
ϕ_1 ϕ_2 ϕ_3 ϕ_4 ϕ_{11}
D Q
S
R
$\overline{\text{Clock}}$
Set

Fig. **4.1. 9-tap Circular Buffer Architecture FIR Filter Block Diagram**

As shown in Fig. 3.5, the flip-flop loop in the clock generator creates eleven clock phases ϕ_1 to ϕ_{11} controlled by the input master clock. Then control signals A_1 to A_{11}, and A_1' to /A_{11}' for the *S/H's* array are created by using a group of nor gate to add the Clear signal. After control signals A's, /A's, CLR, /CLR generated, they go to the eleven Sample and Hold circuits respectively to control each *S/H* tracking input signal and holding sampled values in the proper clock phases. Combining the held values of each *S/H* by multiplexer as described in section 3.3 and Fig. 3.4, the 9-tap input signal samples are ready to be fed into the multiplier circuits.

4.1.2 FIR Filter Design. To test the filtering capabilities of this 9-tap eighth order FIR filter, a notch filter is designed by using the software package MATLAB. Using a sampling frequency of 5MHZ, the notch filter with rejection at 1MHZ was designed. To get the perfect notch filter frequency response, the completely overlapped zeros were designed first. The frequency characteristic is shown in Fig. 4.1. As can be seen, this is almost an ideal notch filter, it presents very low attenuation until the frequency close to the notch point. For obtaining this frequency characteristic, the original weight values generated from MATLAB are:

1.0000, -2.4721, 6.2918, -8.3607, 10.7295, -8.3607, 6.2918, -2.4721, 1.0000 (4.1)

It's obviously a symmetric structure, which means it has linear phase response as shown. However, the above weight values can not be used directly in the designed filter system. The reason is that the multiplier circuits used in this system have the limitation on linear operation range, which is from -2 volts to +2 volts for the input signal. Also, because of the accuracy of the multiplier, only a two and at most three digit decimal fraction is meaningful for the weight values. So, we have to scale and truncate these values before they are used in the system. In this design, we scale the weight values in ±2V range, and first round into two digit decimal fraction when truncating. The new generated weight values are:

0.19, -0.46, 1.17, -1.56, 2, -1.56, 1.17, -0.46, 0.19 (4.2)

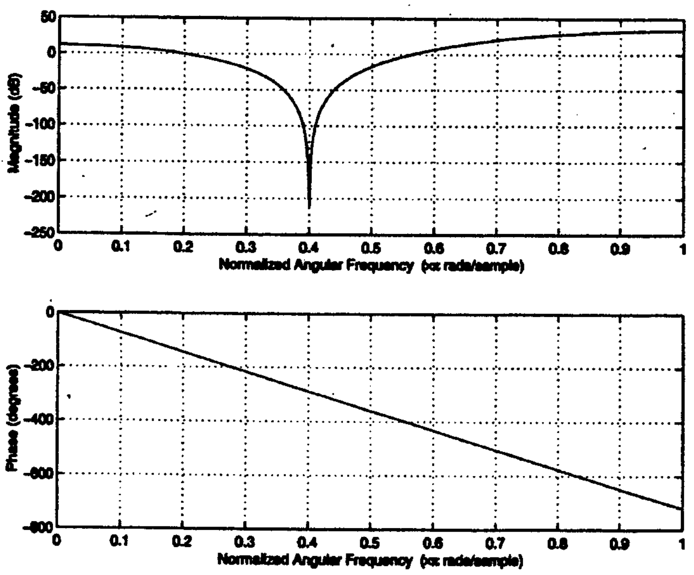


Fig. 4.2 The Ideal Frequency Response

Using these weight values, we can get new frequency response[10]. Obviously, this coefficient scaling and truncating cause a relatively big distortion in frequency characteristic, the central notch point has already move away from the frequency expected. The linear phase property is kept because of the maintenance of symmetric impulse response structure.

If we keep a three digit decimal fraction, the weight values become the following:

0.186, -0.461, 1.173, -1.558, 2, -1.558, 1.173, -0.461, 0.186 (4.3)

Using these weight values, the frequency response shown as in Fig. 4.3 is obtained. The notch point can still be kept at 1MHZ expected frequency point. However, the frequency range for the magnitude is less than -50dB is 0.875 - 1.125MHZ, which is a little bit wider than the ideal one shown in Fig. 4.2. of around 0.9 - 1.1MHZ. It is an acceptable result for the 8-th order notch filter with this kind of architecture.

4.1.3 System Timing Analysis. The system illustrated in Fig. 4.1 is a parallel structure. The new sampled values will be generated each clock cycle in the nine taps simultaneously. Then after multiplication and addition, the final output is generated each clock cycle. The sampling generation is accomplished in parallel with the multiplication/addition. Hence, the minimum clock duration of the system should be the larger value between the sampling phase and the multiplication and addition circuit delay. From Fig. 4.1 we know, the longest signal path circuit delay should be one delay of multiplier

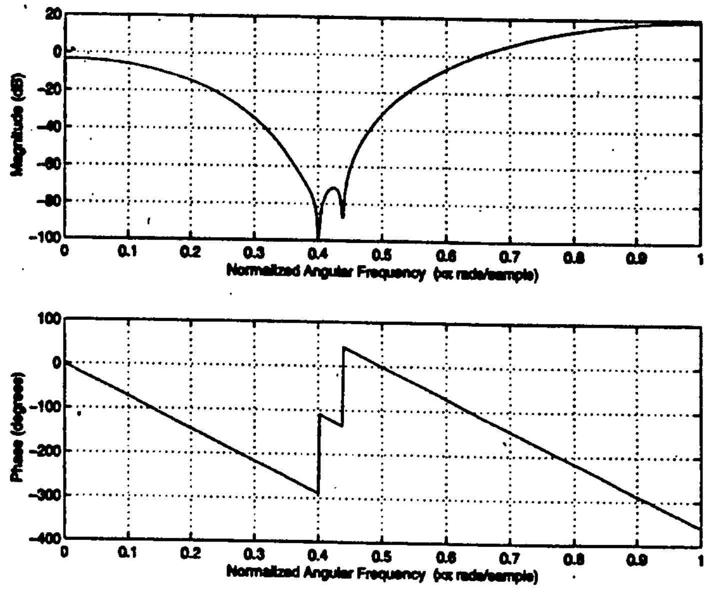


Fig. 4.3 Frequency Response with Three Digit Decimal Fraction

circuit plus four times delay of addition circuit. According to the HSPICE results, the propagation delay of multiplier and addition circuit are 31NS and 5NS respectively. So, the total delay getting from the multiplier and adder circuits is

$$Td = d_m + 4d_a$$
$$= 31NS + 4 \times 5NS = 51NS.$$

This value is much smaller than the needed sampling phase time which is 200 NS. Thus, the whole system clock cycle can be decided by the duration of sampling phase, which is 200 NS in this case. Then, we get the system clock rate of 5MHZ. This reflects the high speed advantage of circular buffer architecture. It can achieve much higher clock rate than delay line structure which has the almost same Sample and Hold circuit, but lower clock rate of 650KHZ [6].

4.2 System Simulation. To test the system function of the designed notch filter, the values of (4.3) are used as filter weights. HSPICE is used to simulate the circuit's behavior, and different frequency input signals are applied to test the frequency selectivity. First, a 100KHZ sine wave with 2V magnitude input signal was used. The output amplitude should be the one sixteenth of the input amplitude because of the 0.5 gain of both multiplier and adder circuit as shown in Fig. 4.1, if we double the first two weight values in (4.3). From Fig. 4.4, we got the peak amplitude RMS value of the 1MHZ output within five periods is 3.28mV. The ratio to 100KHZ output is 0.026. Also, the input signals with frequencies of 250KHZ, 1.75MHZ, and 2MHZ were tested to show the system functionality. These simulation values versus the MATLAB results with the same weights are plotted in Fig. 4.4.

The simulation results reveal some problems of this system. The passband circuit simulation results compared closely with the MATLAB results. However, the attenuation at the notch frequency is less for the circuit simulation. The inaccuracies in the Sample and Hold circuit and the nonlinearity of the multiplier and adder produce some distortion in the filter output. This is in addition to the scaling and truncating of the weight values which already brought some theoretical distortion in the MATLAB result as shown in Fig. 4.3. It is not an ideal notch filter. This is due to the sensitivity of the weight coefficients. It is analogous to the quantization error in the digital filter system. As the filter order is increased, the departure from the ideal response will become worse. The design presented here is restricted to relatively low order filters for applications requiring only moderate accuracy, such as line equalization. An adaptive filter architecture could be realized using the same building blocks with an error correcting system to improve the accuracy. Also reconfiguring the filter architecture to use a Cascade or Parallel realization could improve filter performance.

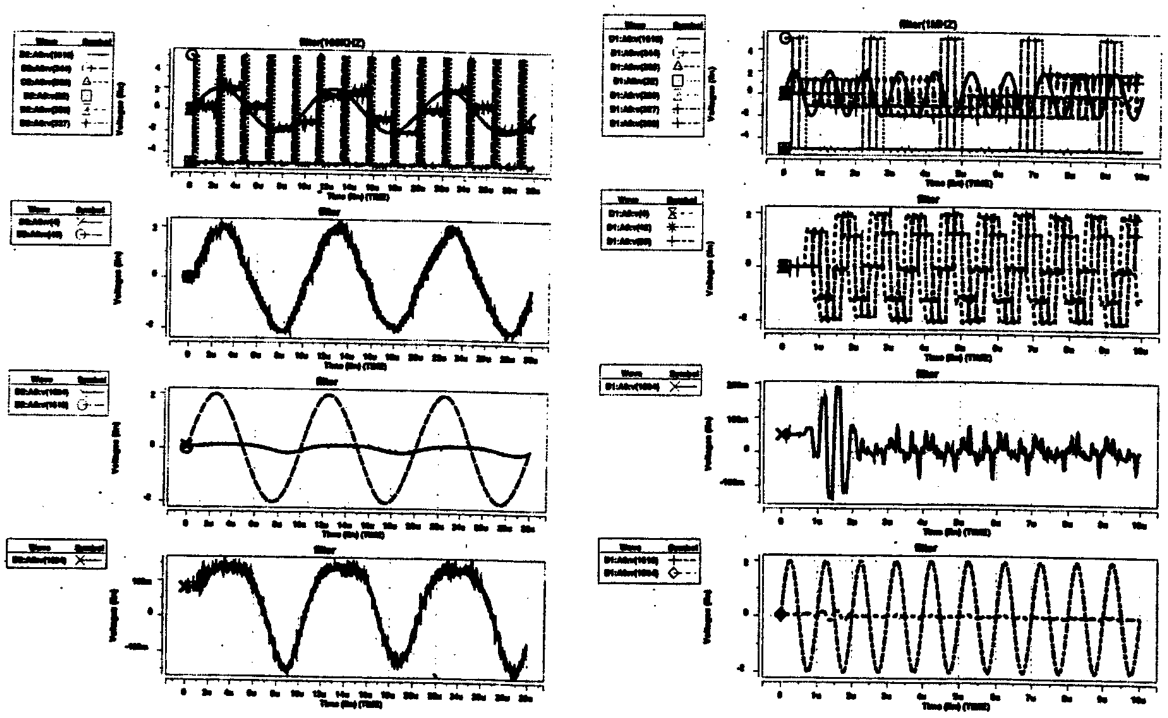


Fig. 4.4 System Simulation Result of 100KHZ and 1MHZ Results VS. MATLAB Results. ×: Test Frequency Point, Solid Line: MATLAB Result, Dash Line: System Simulation Result

5. Conclusions

The modular design and simulations of a 9-tap discrete time analog FIR filter using a circular buffer architecture has been presented. Each building block module in the system is custom designed based on available standard cells, and simulated, resulting in several fully functional analog building blocks to provide the required components for the whole system. The filter is presented in 1.2 micro CMOS technology, with ±5 volts power supplies. The resulting design can be applied to applications requiring moderate speed and accuracy. A CMOS Wide Range Differential Amplifier from the CMOS Analog Library was customized to a voltage follower to build the Sample and Hold module, a key component of the filter system. This module dictates the speed of the whole system. It has good performance based upon simulations and is quite useful for some basic applications. The analog switch circuit based multiplexer module plays a very important role in the system architecture, even though it has relatively simple configuration. Care was more taken on the arrangement of the switch matrix. The *S/H's* held values can be routed to the appropriate multiplier each clock cycle by carefully dealing with the relation between control clock phase and *S/H's* held sampled values. This kind of matrix configuration is very flexible, so it can be suitable for any number of taps or the order of the filter system by just simply increasing or decreasing the number of analog switches. Most important of all in this design is the idea of circular buffer architecture. It is the direct reason for the high speed acquisition. Having the same Sample and hold module, the direct-form (delay-line architecture) FIR filter just achieves 650KHZ maximum clock frequency. However, using circular buffer architecture, at least 5MHZ clock frequency is obtained in this design.

Using the designed modular blocks, the FIR filter system is built and simulated, and filtering capability is demonstrated. However, as previously stated, the filter can only be applied where moderate accuracy and speed is required. Sub-micron technology can be used to make a more area efficient design and lower level voltage source can be applied for less power consumption. These would be obvious future enhancements for the modular FIR design presented.

References

[1] H. Baher, Analog and Digital Signal Processing, John Wiley and sons, 1991.

[2] S. Lyle, G. Worstell, and R. R. Spencer, An Analog Discrete-Time Transversal Filter in 2.0um CMOS, Proc. 26th Annual Asilomar conf. Signals, Systems, and Computers, 1992.

[3] D. Xu, Y. song, and G. T. Uehara, A 200MHZ 9-Tap Analog Equalizer For Magnetic Disk read Channels in 0.6um CMOS, ISSCC, Dig. Tech .Papers, Feb, 1997.

[4] David A. Johns and Ken Martin, Analog Integrated Circuit Design, John Wiley and Sons, 1997.

[5] R. Jacob Baker, Harry W. Li, and David E. Boyce, CMOS Circuit Design, Layout and Simulation, the Institute of Electronics Engineerings, 1998.

[6] Gabriel J. Gomez, CMOS VLSI Implementation of A Discrete-Time Analog Adaptive Filter, M.S. Thesis, Wright State University, 1991.

[7] J. Xu, R. Siferd, R. Ewing, High Performance CMOS Analog Arithmetic Circuits, Journal of Analog Integrated Circuits and Signal Processing, 1999.

[8] J. Sonntagetal, A High Speed, Low Power PRML Read Channel Device, IEEE Trans. Magn.,Vol.31, 1995.

[9] Xiaodong Wang and Richard R. Spencer, A Low Power 170 MHZ Discrete Time Analog FIR Filter, IEEE Journal of Solid State Circuit 33(3), 1998.

[10] Rong Wang, CMOS Analog Implementation of a Discrete-Time 9-Tap FIR Filter with Circular Buffer Architecture, M.S. Thesis, Wright State University, 2000.

TASK 16

Wavelet Transform for Real-Time Video/Audio Communications

Principal Investigator:	Prof. Yuan Zheng
Institution:	Ohio State University, Columbus, Ohio
Period of Performance:	June 22, 1999 to September 1, 2001
Task:	Investigate a lifting scheme to create a number of new integer wavelet transforms with the goal of improving the speed of wavelet computation. The new integer wavelet transforms shall involve only addition, subtraction and shifting of integers. Using the improved-speed wavelet transforms developed in Task 1, develop and implement method(s) for increasing the size of video frames from the present nominal 160x120 pixels to at least 320x240 pixels without decreasing the number of frames transmitted per second, and with no decrease in image quality. Investigate, implement, and evaluate new approaches for video/audio synchronization to achieve simultaneous video/audio communications.

Wavelet Transform for Real-Time Video/Audio Communications

Yuan F. Zheng
Dept. of Electrical Engineering
The Ohio State University
Columbus, OH 43210
Tel: (614) 292-2571
FAX: (614) 292-7596
Email: zheng@ee.eng.ohio-state.edu

1. Introduction

The objective of Task 16 is to study advanced mechanisms for compressing of video signals for real-time communication of multimedia using the Internet or via wireless. We have been using wavelet transform since it can provide higher compression ratio with better video quality. We developed a number of technical approaches to effectively apply the wavelet transform for the compression purpose.

The first approach is related to the computation which is extensive in the wavelet transform. To make the real-time application of wavelet transform possible, it is important to simplify the computation. We have used a so-called **Lifting Scheme** which can convert the wavelet transform from floating-point computation to integer-based computation, and thus reduce the computation time. Furthermore, we developed a Packed computation approach which packs multiple wavelet coefficients into a single computation word. Parallel computation is thus achieved without using multiple processors. The result is up to 37% improvement of the computation speed. The integer computation approach has another advantage, i.e., it is much simpler to implement in a VLSI chip than in floating-point computation. This creates a significant advantage for developing embedded compression systems which will cost significantly less than using the floating-point arithmetic.

The second approach we developed is to combine audio-video compression by applying the wavelet transform uniformly. The idea is to treat the audio signal as a virtual video and combine it with the video frames. As a result, the same wavelet transform package can be applied to both video and audio without developing new compressing software for audio. Likewise, a single application specified integrated circuit can be used for compression of both video and audio. Furthermore, this new approach can improve the compression performance of the audio signal because redundancy is explored not only between neighboring samples but also between samples in distance. This idea is very much like video in not only 2D for intra-frame compression but also 3D for inter-frame compression.

Based on the new developed mechanism, we have developed a software package which can achieve wavelet compression of color video in real-time. Communication software has also been developed which enables us to transmit color video from The Ohio State University in Columbus, OH 43210 to the Air Force Research Laboratory (AFRL) in

Dayton, Ohio as well as to the Information Directorate division of AFRL in Rome, New York.

In the rest of this report, we will present the details of the two research activities, i.e., Packed Integer Wavelet Transform Constructed by Lifting Scheme in Section 2, and 2-D Combined Audio-Video Compression in Section 3. References for each section are separately listed at the end of each section.

2. Packed Integer Wavelet Transform Constructed by Lifting Scheme

2.1 Introduction

The wavelet transform has received much attention in the field of image compression [1-3]. It provides great potential of achieving better Rate-Distortion performance than conventional DCT-based approach (e.g. JEPG). One problem associated with the wavelet image compression technology is the high computational complexity. Although floating point arithmetic is nearly as fast as integer arithmetic when their operands have the same data length, the integer wavelet transform can be implemented much faster than the floating point wavelet transform in almost all general purpose computers because the floating point wavelet transform demands for longer data length than the integer wavelet transform does. Another benefit of using integer wavelets is the reversibility. That is, the image can be reconstructed losslessly because all the coefficients are integers and can be stored without rounding-off errors. The lifting scheme is a new method for constructing integer wavelet transform [4]. Recently, biorthogonal wavelets constructed by the lifting scheme have been identified as very promising filters for lossless/lossy image compression applications [3], [5]. By making use of similarities between the high and low pass filters, the lifting scheme reduces the computation complexity by a factor of two compared with traditional wavelet transform algorithms. With certain modifications, the corresponding wavelet transform can even be calculated with only integer addition and shift operations that make the computation even faster [3]. Besides, the transform is reversible which means that it can be used for both lossless and lossy image compression. Furthermore, the inverse wavelet transform can be immediately found by undoing the operations of the forward transform.

Modern wavelet-based image compression systems [1], [2] contain three building elements: (1) wavelet transform, (2) successive quantization, and (3) adaptive entropy coding. Typically, more than 60% of the time used in image compression is consumed by the wavelet transform. It is very crucial to speed up the computation of the wavelet transform for real-time image and video compression applications, especially for large scale and color images. While integer wavelets using the lifting scheme significantly reduce the computation time [5], we propose a new approach to further speed up the computation of the wavelet transform.

The method is based on the fact that the 16-bit integer arithmetic has the same speed as the 32-bit integer arithmetic in contemporary computers while a 16-bit data unit is sufficient for most integer wavelets. We can therefore pack multiple pixels (wavelet coefficients) in a single long word during the computation of the reversible wavelet transform. As a result, operations on multiple pixels (wavelet coefficients) can be performed at once. Thus, the computation time as well as the working memory space can be dramatically reduced. Furthermore, we observed that for reversible integer wavelets constructed by the lifting scheme, if the dynamic range of the coefficients is within $[-2^{15}; 2^{15}-1]$, their corresponding packed version is also a reversible transform. Consequently, the quality of the reconstructed images is the same as an unpacked transform method. Performing two logical arithmetic operations with one physical operation was proposed earlier for DCT based JPEG compression and decompression ([6]). We use the same approach, but apply it to the integer wavelet transform that needs different considerations in the design of multiple arithmetic operations and analysis of errors.

2.2 Integer Wavelet Transform

The wavelet transform can be considered as a subband transform and implemented with A filter bank [7]. Figure 2.1 describes the general block scheme of a one-dimensional biorthogonal wavelet transform. The forward transform uses two analysis filters: $\tilde{h}$ (low-pass) and $\tilde{g}$ (high-pass) followed by subsampling, while the inverse transform first upsamples and then uses two synthesis filters: h (low-pass) and g (high-pass). The outputs of the synthesis filters are added together to reconstruct the original signal. The conditions for perfect reconstruction are given by [8]

$$h(z)\tilde{h}(z^{-1})+g(z)\tilde{g}(z^{-1})=2$$
$$h(z)\tilde{h}(-z^{-1})+g(z)\tilde{g}(-z^{-1})=0$$

When $\tilde{h}=h$ and $\tilde{g}=g$, $\{\tilde{h},\tilde{g},h,g\}$ forms an orthogonal wavelet transform.

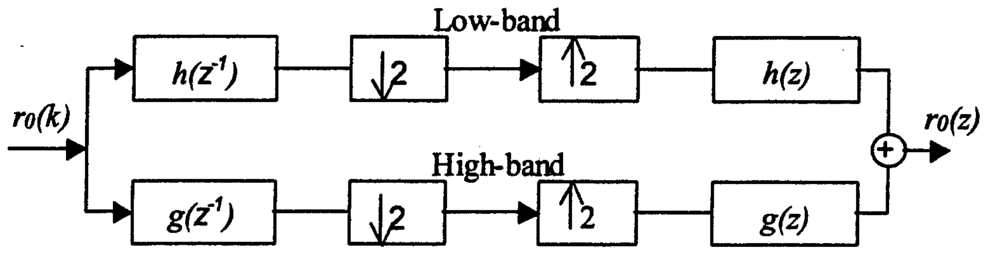


Figure 2.1 Basic filter bank for biorthogonal wavelet transform

By using the polyphase representation of a filter h: $h(z) = h_e(z^2 + z^{-1}h_o(z^2)$, where $h(z) =$

165

$\sum_k h_{2k} z^{-k}$ contains even coefficients, and $h_{o(z)} = \sum_k h_{2k-1} z^{-k}$ contains the odd coefficients, we can assemble the *polyphase matrix P (z)* [5] to represent the filter pair $(\tilde{h}, \tilde{g})$:

$$P(z) = \begin{bmatrix} \tilde{h}_e(z) & \tilde{h}_o(z) \\ \tilde{g}(z) & \tilde{g}(z) \end{bmatrix}$$

Let x(z) =*r0(z)*. The corresponding wavelet transform and subsampling in Figure 2.1 can be written as

$$\begin{bmatrix} r_1(z) \\ d_1(z) \end{bmatrix} = P(z) \begin{bmatrix} x_e(z) \\ z^{-1} x_0(z) \end{bmatrix}$$

where $x_e(z_o)$ and $x_o(z)$ are the even and odd components of $x(z)$.

The lifting scheme [4], [5], provides a new approach for construction of biorthogonal wavelets with compact support. It is proved [5] that any polyphase matrix *p(z)* representing wavelet transform with finite filters can be obtained by performing Lazy wavelet transform followed by alternation *primal* and/or *dual lifting* steps.

With *primal lifting*, starting from two complementary finite filters $\tilde{h}$ and $\tilde{g}$, a new finite filter $\tilde{h}^{new}$ complementary to $\tilde{g}$ is created:

$$\tilde{h}^{new}(z) = \tilde{h}(z) + s(z^2)\tilde{g}(z) \qquad (1)$$

where *s*(*z*) is a Laurent polynomial. Using the polyphase representation, (1) can be written as

$$P^{new}(z) = \begin{bmatrix} 1 & s(z) \\ 0 & 1 \end{bmatrix} P(z)$$

With *dual lifting*, starting from $\tilde{h}$ and $\tilde{g}$, a new finite filter $\tilde{g}^{new}$ complementary to $\tilde{h}$ is created.

$$\tilde{g}^{new}(z) = \tilde{g}(z) + t(z_2)\tilde{h}(z), \qquad (2)$$

where *t*(*z*) is a Laurent polynomial. Using the polyphase representation, (2) can be written as

$$P^{new}(z) = \begin{bmatrix} 1 & s(z) \\ t(z) & 1 \end{bmatrix} P(z)$$

By iteratively repeating this process, *P*(*z*) can be factored into a product of unit upper and lower triangular 2 x 2 matrices, and a diagonal normalization matrix [5]. Alternatively, one can obtain any wavelet transform *P*(*z*) by starting with a simple wavelet transform represented by the diagonal normalization matrix and using the lifting scheme. One of the most interesting advantages of the lifting scheme is that it can be used to create integer wavelet transforms.

It has been proved that if the decomposition and reconstruction by filters $\{\tilde{h}, \tilde{g}, h, g\}$ can be accomplished with only integer arithmetic, we can also create a corresponding integer wavelet transform by filters $\{\tilde{h}^{new}, \tilde{g}, h, g^{new}\}$ which are generated based on $\{\tilde{h}, \tilde{g}, h, g\}$ and $\{s, t\}$. In fact, one can, in each lifting step, round off the result of the filter right before the addition or subtraction. Thus, the forward and inverse integer wavelet transforms are as follows.

Forward Transform

1. Classical subband splitting by applying the analysis filters $\tilde{h}, \tilde{g}$ to $r_0(n)$, the corresponding low-pass and high-pass subbands are $r_1^0(k)$ and $d_1^0(k)$, respectively.

2. Updating the low subband $r_1^0(k)$ by applying the s filter on the high subband $d_1(k) = d_1^0(k)$.

$$r_1(k) = r_1^0(k) + Int\left(\sum_n d_1(k-n)s(n)\right) \qquad (3)$$

or, updating the high subband $d_1^0(k)$ by applying the t filter on the low subband $r_1(k) = r_1^0(k)$.

$$d_1(k) = d_1^0(k) + Int\left(\sum_n d_1(k-n)t(n)\right) \qquad (4)$$

Inverse Transform:

1 Undo the primal lifting with $d_1^0(k) = d_1(k)$, or undo the dual lifting with $r_1(k) = r_1(k)$. This is exactly the "backward" version of (3) and (4). In practice, this comes down to simply changing each + into -, and vice versa.

2 Inverse transform using the synthesis filters, *(h, g)* on the low-pass and high-pass subbands $r_1^0(k)$ and $d_1^0(k)$, and get back the original signal $r_0(n)$.

Notice that if we carefully choose the parameters *{s, t}* in (3) and (4) so that only integer addition, subtraction and shift operations are required in the computation, the wavelet transform can be performed directly by integer arithmetic [3]. For example, for one of the biorthogonal filter banks for image compression, the (2, 6) wavelet that corresponds to the TS transform [2], [9],

$$\tilde{h} + \frac{1}{\sqrt{2}}(1+z),$$

$$\tilde{g} = \frac{1}{8\sqrt{2}}(-z^{-2} - z^{-1} + 8 - 8z + z^2 + z^3)$$

the decomposition can be done via the following lifting steps [3]:

$$d_1^0(k) + r_0(2k) - r_0(2k-1),$$

$$r_1(k) = Int\ (\frac{d_1^0(k)}{2}) + r_0(2k+1),$$

$$d_1(k) = Int(\frac{r_1(k-1) - r_1(k+1)}{4} - d_1^0(k).$$

Its reconstruction algorithm immediately follows as we undo the decomposition operations. Based on the work of [3] and [5], several other integer wavelet transforms, such as S transform, (5,3) transform, and (S + P) transform, etc., can be modified and only need integer addition, subtraction and shift operations. This property gives rise to the possibility of our proposed packed reversible integer wavelet transform, which takes advantage of the 32-bit or 64-bit computational capability of modern computers.

2.3 Packed Integer Wavelet Transforms

The basic idea of packed integer wavelet transform is to pack multiple pixels (wavelet coefficients) in one integer word. Therefore, multiple additions/subtractions can be accomplished in one instruction cycle. There are two issues associated with the packed approach. One is overflow of the magnitude, and the other is carry over of the sign bit. When either occurs, the result of the transform will be affected. It can be shown that overflow is not a concern in most applications of the wavelet transform.

The packed addition and subtraction will not overflow if the wavelet transform is limited to a few levels of the multi-band decomposition. This is because in the wavelet transform, the low-pass filters $h(n)$ and $\tilde{h}(n)$ must satisfy [10].

$$\sum_n h(n) = \sum_n \tilde{h}(n) = \sqrt{2}.$$

As a result, the magnitude of the coefficient is increased by $\sqrt{2}$ for every level of decomposition. For the 2D case, the increase will be 2. This will enlarge the range of the coefficient by 1 bit in each level of decomposition. If the decomposition is limited to 3 to 4 levels, additional 3 to 4 bits will be sufficient to hold the dynamic range of the coefficients. The above is based on a condition that every $h(n)$ or $\tilde{h}(n)$ is positive. If the coefficients are not all positive, one may have $\Sigma_n |h(n)|$ or $\Sigma_n |\tilde{h}(n)|$ be greater than $\sqrt{2}$. And the dynamic range may be increased by more than 1 bit in each level of decomposition. In reality, however, it is still not much greater than $\sqrt{2}$. For example, in the S, TS, (5,3) and (S+P) transforms, which we have implemented using the integer computation, the limit is $\frac{2.5}{\sqrt{2}}$. Consequently, 4 levels of decomposition will introduce at most 7 bits in the worst case. If the pixels are assigned with 8 bits in the first place, the dynamic range will not exceed 16 bits.

In the integer computation, the magnitude does not even increase (see the lifting steps shown in the previous section). Reference [3] also confirms this property and calls is the *Precision Preservations* property. Consequently, overflow of the magnitude is not a concern in our applications. On the other hand, the most significant bit of a binary number in the computation is the sign bit; when multiple numbers are packed in a long word, the carry over of the sign bit from the low word to the high word will alternate the magnitude of the high word. This issue needs a careful consideration and will be discussed in the following sub-section.

A. Packed Computation

Let $A = (a_{n-1}, \ldots, a_1, a_0)$ and $B = (b_{n-1}, \ldots, b_1, b_0)$ be the packed integer of $a_i, b_i \in [-2^{15}, 2^{15}-1]$ $(I=0, \ldots, n-1)$. Notice that for 32-bit computers, $n = 2$; for *64* bit computers, $n = 3$, etc. The packed addition and subtraction of a_i and b_i are denoted as $S = A + B$ and $D = A - B$, respectively, where $S = (s_{n-1}, \ldots, s_1, s_0)$, $D = (d_{n-1}, \ldots, d_1, d_0)$, $s_i, d_i \in Z$. We observe that even when $s_{n-1}, d_{i-1} \in [-2^{15}, 2^{15}-1]$, it is not necessary that $a_i + b_i = s_i$ or $a_i - b_i = d_i$. For example, let $A = (4, -3)$, and $B = (5, 4)$. We have $S = (10, 2)$ instead of $(9, 2)$. Although $s_0 = 2$ does not overflow, there is a carry bit generated by the low word addition that is propagated to the high word. As a result, $s_1 = 10 \neq 4 + 5$. Similar situations exist for the packed subtraction. For $A = (111, 72)$ and $B = (108, 82)$, we have $D = (2, -10)$, where $d_i = 2 \neq 111-108$.

Although the carry over of the sign bit alters the result of the packed addition and subtraction, the effect is very small (limited to ± 1 for each level of wavelet transform). Consequently, we have the following possibilities for s_i and d_i $(0 \leq i \leq n-1)$:

$$s_i \;\; = a_1 + b_i \quad if \quad s_{i-1} a_{i-1} b_{i-1} \geq 0 \, or \, i = 0 \\ a_i + b_i + 1 \quad if \quad s_{i-1} a_{i-1} b_{i-1} < 0 \tag{6}$$

$$d_i = \;\; a_i - b_i \quad if \, d_{i-1} a_{i-1} b_{i-1} \geq 0 \; or \; 1 = 0 \\ a_i - b_i - 1 \quad if \quad d_{i-1} a_{i-1} b_{i-1} < 0 \tag{7}$$

In addition to the packed addition and subtraction operations, we further introduce the packed shift operation to avoid the interference on a_{i-1} from a_i when we apply a right shift to the packed integer A (i.e., divided by 2^l where l is the number of shifts). The functionality of the packed shift operation is equivalent to that of the following operations: $\{Int\,(a_i/2^l)\}$, $I = 0, \ldots, n-1$.

B. Packed Integer Wavelet Transform

In general, multiple coefficients can be packed into one integer provided that the width of the data-path is sufficient. Since 32-bit is a typical data width for the state-of-the-art computers, we will focus on packing two pixels/coefficients into one 32-bit integer in our implementation. This does not mean that our algorithms are limited to 32-bit machines.

The general algorithm for the packed integer wavelet decomposition can be described as follows.

1. Pack two rows/columns.
Let $r_0(2n, k)$ and $r_0(2n+1, k)$ be the $(2n)^{th}$ and $(2n+1)^{th}$ row or column, respectively. Suppose the packed result is saved in $\overline{r_o(k)}$, we have

$$\overline{r_o(k)} = (r_0(2n, k), r_0(2n + 1, k)).$$

2. Complete the packed wavelet transform using the simple filters $\tilde{h}$ and $\tilde{g}$. The algorithms are the same as (3) (4) except that we use $\overline{r_o(n)}$ as the input instead of $r_0(n)$. The intermediate results $\{r_1^0(k), d_1^0(k)\}$ and the final results $\{r_1(k), d_1(k)\}$ are also in packed format.

3. Unpack two rows/columns

$$r_i(2n,k)=HW(\overline{r_1(k)}), \qquad r_1(2n+1,k)=LW(\overline{r_1(k)}),$$
$$d_1(2n,k)=HW(\overline{d_1(k)}), \qquad d_i(2n+1,k)=LW(\overline{d_1(k)})$$

where $HW(.)$ and $LW(.)$ represent the high word and low word of the packed integer, respectively.

Similarly, the algorithm for packed wavelet reconstruction can be described as follows.

1. Pack two rows/columns

$$\overline{r_1(k)}=(r_1(2n,k), r_1(2n+1,k)),$$
$$\overline{d_1(k)}=(d_1(k), d_1(2n+1,k)).$$

2. Undo the packed primal and/or dual lifting. Notice that the reconstructed signal is in packed format $r_0(k)$.

3. Unpack two rows/columns

$$r_0(2n,k) = HW(r_0(k)), \qquad r_0(2n+1,k) = LW(r_0(k)).$$

C. Performance Analysis

Although we may introduce a ± 1 difference in each packed addition or subtraction operation, the impact of this difference on image compression and reconstruction is rather negligible because: (1) It is not necessary that every packed addition or subtraction will contribute a $+1$ or -1 to s_i or d_i; (2) According to (6) (7), alternating addition and subtraction can cancel out the 1 bit difference; (3) Right shift operations can further reduce the effect of bit propagation.

Consider applying the TS transform to the *Peppers* image. In a *3*-level packed and unpacked TS transform, the maximum difference is only *3* as expected and its population is limited to *0.58%* of the entire coefficient set. The majority of the coefficients is the same *(52.24%)* or the difference is only *1 (41.38%)*. The difference becomes even smaller and negligible after the scalar quantization, where *98.60%* of the coefficients are the same and the difference for the remaining *1.4%* is limited to *1*.

According to [5], every wavelet with finite filters can be obtained as the Lazy wavelet followed by a finite number of lifting steps and scaling. The Lazy wavelet is reversible for either packed or unpacked data because it does nothing but splitting the original signal into even and odd indexed samples. The lifting step is also reversible because the packed reconstruction is the exact reverse of the packed decomposition provided that there is no overflow. As a result, the entire packed wavelet transform is reversible when the dynamic range of its coefficients is within $[-2^{15}, 2^{15} - 1]$ which is true for most applications.

In our scheme, we have to unpack each row before we can pack the columns, and vice versa. This process will induce some overhead. However, in the practical implementation, pack and unpack operations can be done simultaneously with memory copy which is also required by the original wavelet transform to move data between the image matrix and the working buffer for the wavelet transform. So the overhead of the pack/unpack operations is negligible.

2.4 Experiment Results

To verify the advantage of the packed integer wavelet transform, we compared its performance with that of the original integer wavelet transform for image compression. The coding algorithm used was a three-level wavelet transform, followed by a scalar quantization and stack-run coding [11]. No further entropy coding was used after the stack-run coding. The step-size used in the quantization is 16. We implemented four different integer wavelet transform algorithms (S, TS, (5, 3), and (S+P)) along with their packed versions. This experiment was done on a 166 MHz Pentium PC with 16 MB memory. We compared the performance of the packed and unpacked integer wavelet transforms on four images: *Girl* (256 x 256), *Lena* (512 x 512 x 3), and *Man* (1024 x 1024), respectively. Table 1 shows the compression ratio for both the packed and unpacked computations in terms of the bit-rate. One can see that the difference between the packed and unpacked transforms is very small. Table 2 shows the decomposition time savings and reconstructed image quality for the four packed wavelets vs. the original ones. We can see that up to 37% savings in the decomposition time can be achieved by using our packed transform algorithms. We also observed that the speed-up factor is nearly image invariant and wavelet-type invariant. Since the packed wavelet transform is symmetric in terms of computation, it follows that we have very close performance in the reconstruction time as in the decomposition time. In the above experiment, the maximum difference in compression ratio between the packed and unpacked transforms is less that 0.25 and the difference in the reconstructed image quality (PSNR) is limited to 0.24 dB.

2.5 Conclusions

In this chapter, an efficient mechanism for computing the integer wavelet transform, the packed integer wavelet transform, is described. The proposed packed transform can speed up the decomposition/reconstruction process up to 37 percent with a comparable performance in the compression ratio and reconstructed image quality. This approach is quite suitable for the applications in which the speed is a critical factor but no additional hardware such as vector processing devices or other embedded system is available. In reality, many types of special-purpose processors exist which can significantly speed up the computation of the wavelet transform. The current approach is not to replace them but to provide an alternative, especially when the extra hardware is not available to the users.

Although the current implementation of our proposed algorithms is based on 32-bit computers, the packed wavelet transform algorithm is not limited to 32-bit data-path. More speed-up can be achieved if the software is tested on 64 bit machines, or future computers with even wider data-path. The speed-up mechanism is made possible by using the lifting scheme. It is shown in [5] that the cost of the lifting algorithm for computing the wavelet transform is only half of the cost of the standard algorithm. For certain wavelet transforms, the lifting scheme can result in only addition, subtraction and shifting of integers. By using the packed integer wavelet transform, the computation cost can be further reduced. Finally, it should be pointed out that the current approach is not suitable for completely lossless image compression because it introduces a difference during the process of computation, although the magnitude of the difference is very small.

Table 1
DIFFERENCE IN COMPRESSION RATIO

		Girl 256 x 256	Lena 512 x 512	Peppers 512x512x3	Man 102 x 1024
S transform	unpacked	19.70	17.48	35.52	16.94
	packed	19.55	17.42	35.31	16.82
TS transform	unpacked	24.20	23.07	48.37	21.66
	packed	23.97	22.98	47.72	21.44
(5, 3) transform	unpacked	22.18	21.69	46.43	19.13
	packed	22.25	21.65	46.27	19.09
(S + P) transform	unpacked	26.48	25.86	52.20	23.63
	packed	26.12	25.78	51.81	23.38

Table 2
COMPARISON OF DECOMPOSITION TIME (MSEC) AND RECONSTRUCTED IMAGE QUALITY IN PSNR (DB)

		Girl 256 x 256		Lena 512 x 512		Peppers 512x512x3		Man 102 x 1024	
		time	psnr	time	psnr	time	psnr	time	psnr
S transform	unpacked	46	29.70	226	29.80	318	28.45	978	28.89
	packed	33	29.94	150	30.06	213	28.68	674	29.08

TS transform	unpacked	51	30.74	246	30.96	348	29.57	1061	29.80
	packed	35	30.76	156	30.96	222	29.59	712	29.80
(5, 3) transform	unpacked	62	32.05	268	32.39	397	30.41	1103	30.87
	packed	42	31.86	176	32.18	257	30.38	811	30.76
(S + P) transform	unpacked	52	30.71	251	30.90	357	29.43	1063	29.63
	packed	36	30.76	165	30.90	238	29.41	734	29.62

2. 6 References

1. A. Said and W. Pearlman, "A new fast and efficient image codec based on set partitioning in hierarchical tree," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 243-250, 1996.

2. E. Schwartz, A. Zandi, and M. Boliek, "Implementation of compression with reversible embedded wavelets," in *Proceedings of SPIE 40th Annual Meeting*, San Diego, CA, 1995.

3. H. Chao and P. Fisher, "An approach to fast integer reversible wavelet transforms for image compression," http://www.compsci.com/wchao/Publication/Pub_No.1.ps.gz.

4. W. Sweldens, "The lifting scheme: a custom-design construction of biorthogonal wavelets," *Applied and Computational Harmonic Analysis*, vol. 3, no. 2, pp. 186-200, 1996.

5. I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *Journal of Fourier Analysis and Applications,* 4 (no. 3), pp. 247-269, 1998.

6. J. Allen, "An approach to fast transform coding in software," *Signal Processing: Image Communication,* vol. 8, pp. 3-11, 1996.

7. S. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp.674-693, 1989.

8. I. Daubechies, "Ten lectures on wavelets," in *CBMS-NSF Regional Conference Series in Applied Mathematics, SIAM,* Philadelphia, 1992.

9. J. Vallasenor, B. Belzer, and J. Liao, "Wavelet filter evaluation for image compression," *IEEE Transactions on Image Processing*, vol. 4, no. 8, pp. 1053-1060, 1995.

10. R. Gopinath, C. Burrus, *Introduction to Wavelets and Wavelet Transform, A Primer*, Prentice Hall, Upper Saddle River, NJ, 1998.

11. M. Tsai, J. Villasenor, and F. Chen, "Stack-run image coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 5, pp. 519-521, 1996.

3. A Novel 2-D Combined Audio-Video Wavelet Compression and Its Application in Multimedia Synchronization

3.1 Introduction

As the advent of new network technologies, applications of the Internet are becoming more and more popular, such as videoconferencing, audio and video broadcasting, and video on demand (VOD), etc. There are two major problems experienced by these applications. One is the conflict between the huge amount of multimedia data and the limited bandwidth of the Internet; the other is synchronization between multimedia such as video and audio. A lot of research has been conducted to solve these problems. Various compression algorithms are developed [1, 2, 3] to reduce the required bandwidth, and different time control methods are proposed to keep multimedia synchronized [6, 7]. However, not much effort has been made to solve the two problems simultaneously. Typically, different compression methods are applied to different media (video and audio in most cases), and a multiplexing protocol is used to schedule the transmission of different media [8]. In general, different media are captured, compressed and transmitted separately so that additional control models must be designed to achieve inter-media synchronization. Thus, a transmission system is often complex and the multiplexing scheme is not efficient in the usage of the Internet bandwidth. We proposed a novel two-dimensional (2-D) combined video-audio compression scheme using the integer wavelet transform, which performs very well at low bit rate. Based on this novel compression scheme, we designed an integrated adaptive multimedia transmission system, which can achieve easy synchronization.

3.2 Combined Audio-Video Compression

A good compression algorithm for real-time applications must have a high computation speed, maintain a high quality video, and use low bit rate. Since the wavelet transform has better performance than the DCT based MPEG approach, we choose the wavelet transform for our applications. The computation of the wavelet transform is relatively complex, which has given rise to many studies for speeding up the computation. A special integer-based wavelet transform called ***Packed Integer Wavelet Transform*** was developed by our research group earlier [4], which improves the computation speed significantly while maintaining a comparable performance in comparison with ordinary integer-based wavelet transform. As the packed approach is used in our study, it is briefly introduced below.

A. Packed Integer Wavelet Transform

In comparison with the orthogonal wavelet transform, biorthogonal wavelet transform can be computed much faster because biorthogonal wavelets have shorter filter coefficients than orthogonal ones when both have the same number of vanishing moments [5]. The basic biorthogonal wavelet decomposition scheme is described in Figure 3.1. Here the coefficients in the jth level (A_j) are simultaneously decomposed into the (j+1)th level of approximation (A_{j+1}) and detail (D_{j+1}) coefficients using the low-pass and high-pass impulse response $\tilde{h}(z)$ and $\tilde{g}(z)$, respectively. This binary-tree decomposition can be easily extended to 2D signals.

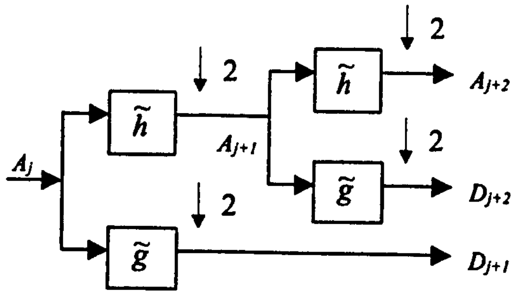


Figure 3.1 Wavelet transform for multi-resolution decomposition

The procedure of reconstruction is the inverse of the wavelet decomposition as shown in Figure 3.1. However, the synthesis filters $h(z)$ and $g(z)$ are different from the analysis filters $\tilde{h}(z)$ and $\tilde{g}(z)$. The filters must satisfy the following equations:

$$\tilde{h}(z)h(z)+\tilde{g}(z)g(z) = 2 \qquad (1)$$
$$\tilde{h}(z)h(-z)+\tilde{g}(z)g(-z)=0 \qquad (2)$$

Integer-based wavelet transform is faster than floating-point wavelet transform because integer computation consumes much less time by the computer. In addition, integer wavelet can achieve better quality because no round-off errors exist in the integer computation. To further improve the computation speed, multiple wavelet coefficients are packed together in one word. This is because modern computer architecture has more than 32 bits in a single word, while one needs only 8-bits to represent a pixel or a wavelet coefficient in the integer-based wavelet transform. In this way, the computation is more efficient because additions or subtractions on multiple coefficients can be performed in a single instruction cycle. In practice, we pack two adjacent rows of a video frame together. It is equivalent to reducing the frame size by one-half. As a result, the packed approach reduces the computation time by 37%.

B. 2-D Combined Media Compression Scheme

The packed integer wavelet transform enables us to achieve high-speed compression. However, in a multimedia transmission system, an important aspect is synchronization. If different media are compressed and transmitted independently, one cannot guarantee their arrival at the client side simultaneously. A time-control scheme must be added to achieve synchronization between different media. Current schemes have two drawbacks. One is that a synchronization scheme always causes a delay because audio and video signals arrive separately, and the early arrival has to wait for the later one. The other is that the synchronization scheme itself requires additional computation. If one can put the audio and video together and treat them as a ***virtually single medium***, there is no need to develop a synchronization scheme. To do so, we also need a unified compression algorithm to process the single medium. That is, both audio and video will be treated as the same kind of signal and be compressed simultaneously. Our answer to this requirement is the novel 2-D combined media compression scheme.

B.1 2-D Audio Compression

A very important step of the 2-D compression scheme is to pack the 1-D audio into a 2-D array just like an image. We call it ***audio image***. Figure 3.2 shows the waveform of a section of speech and its audio image. Just like a normal image, an audio image can be compressed using the integer wavelet transform.

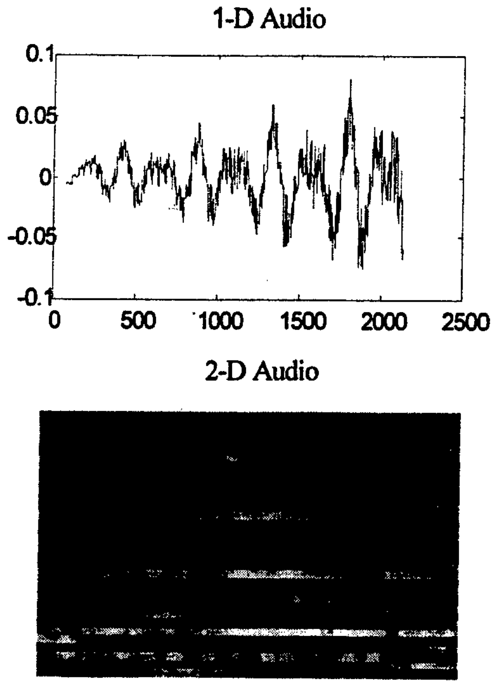


Figure 3.2. 1-D and 2-D representation of an audio signal

Mathematically, the wavelet transform is a good tool to detect correlation between adjacent signals for both audio and video. Since compression is actually achieved by exploiting the correlation, the wavelet transform is more efficient for the audio signal if it is converted into 2-D, because the correlation is found not only between adjacent simples of the signal but also periodically between segments of the signal. In this regard, the 2-D audio compression is similar to the 3-D video compression, in which correlation between video frames is further exploited. Thus the new approach can increase the compression ratio significantly.

Through our research on the 2-D audio compression, we have obtained some interesting results. If the audio signal is very short (less than 1-2 seconds), the 2-D wavelet compression method shows no obvious advantage over the 1-D method. However, if the audio signal is long, which is true for most applications, the performance of the 2-D method is significantly better than 1-D. The following table compares the results of the 1-D and 2-D compression methods on an 8-second long audio signal. The waveform of the original audio signal and the reconstructed signals by the 1-D and 2-D methods are shown in Figure 3.3. In this comparison, both methods apply the Haar wavelet for three decomposition levels.

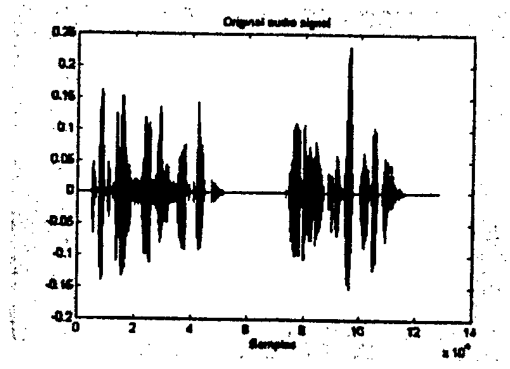
Table 1: Compression performance comparison

	2-D	1-D
Comp. Ratio	10	7.4
Retained energy	94.45%	93.99%

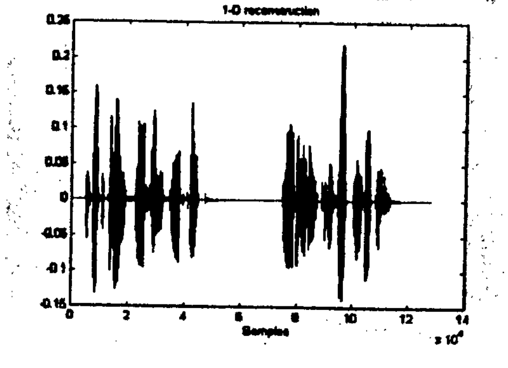
In the table1, the retained energy is calculated by the following formula:

$$retained \quad energy = (1 - \frac{\sum |\hat{S}_i - S_i|^2}{\sum |S_i|^2}) * 100\%$$

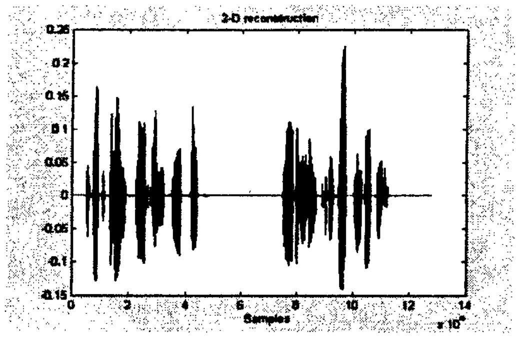
where S_i denotes the original audio signal, and $\hat{S}_i$ denotes the reconstructed signal.



(a)



(b)



(c)

Figure 3.3 Waveform of the original (a), 1-D reconstructed (b), and 2-D reconstructed (c) audio signal

While the waveforms shown are all close to the original signal, the numbers shown in the Table clearly tells the difference between the two compression methods. From the comparison, we can see that the 2-D compression method has a higher compression ratio while achieving a better quality as the method retains more energy. In general, audio signals have strong short-term correlation and some long-term correlation. If an audio signal is cut to proper length, the 2-D method can exploit the long-term correlation and improve the compression ratio.

B.2 Combined Media Compression Procedure

We can take advantage of this new audio compression method to form a uniform compression scheme for both audio and video. This combined media compression method is very useful in multimedia synchronization because the related audio and video signals are transmitted in a single package instead of two separated packages, which may arrive at the client side at random times.

The whole compression procedure includes the following steps. Firstly, the audio and video signals are captured and put into a buffer. Note that the audio and video signals are recorded in parallel and are synchronized at this point. Secondly, the 1-D audio signal is transferred to a 2D audio image with the same width of the video frame. The audio image represents the audio signal, which takes place in the time period when the video frame is recorded. Thirdly, the audio image is attached to the video frame to form a combined frame. Now the combined frame carries both audio and video signals. Fourthly, the packed integer wavelet transform is applied to the combined frame. Since the wavelet transform can maintain the relative locations of the original signals, we can distinguish the coefficients of audio signal from those of the video signal. Finally, we apply different thresholds to achieve different QoS of the two signals. After thresholding, quantization and run length coding are used to achieve compression. Figure 3.4 shows the block-diagram of the compression procedure.

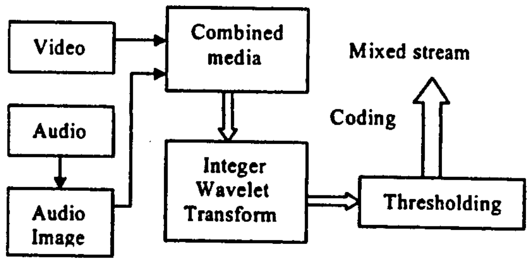


Figure 3.4. Procedure for the combined media compression

After the coding, the original combined frame is converted into a bit stream of a much smaller size and is ready to be sent out via the Internet.

3.3 Testing of the Combined Scheme

To test the performance of the proposed 2-D combined media compression scheme, we designed a demo multimedia transmission system.

A. Structure of the Demo System

The demo system is to capture, compress and transmit live video and audio synchronously. At the server side, there are three functional blocks: data block (recording video and audio), compression block, and transmission block. On the client side, there are feedback, decompression, and playback blocks. In the data block, a DT3152 image grabber and a camera are set to capture the video signal, while a microphone is used to capture the audio signal. After the data capturing, a 2-D combined media frame is formed and put into the 2-D integer wavelet codec. Since the 2-D combined media compression scheme can compress video and audio together and produce a single bit stream, we adopt a single channel transmission scheme. In this way, the mixed audio and video code is transmitted together, and the related audio and video signals can be recovered properly at the client side. Hence synchronization becomes a simple issue.

B. Performance Analysis

Figure 3.5 compares the display windows between the server and client. The window at the server side shows the combined frame (audio is packed at the bottom of the frame) before compression, and the window at the client side shows the reconstructed combined frame. The compression ratio is 10:1 for the video, and 2:1 for the audio. On site evaluation shows that the quality of both video and audio at the client side is very good which is close to the server side, and the media (both video and audio) playback is continuous and synchronous. This experimental result demonstrates that the 2-D combined media compression scheme is successful in achieving synchronization as well as compression.

3.4 Conclusions

We have presented a novel 2-D combined audio-video compression method, which is suitable for real time applications in the Internet. The new method has several advantages over the conventional methods. It simplifies synchronization between multimedia and achieves higher compression ratio for audio. Another advantage, which was not mentioned earlier, is that the audio and video signals are transparent to the compression software because both are treated as the same 2-D "images". As a result, no new software needs to be developed for audio once it becomes available for video. Although only video and audio are mentioned, the method can be extended to other media such as graphics.

3.5 References

1. ISO/IEC 11172-2. Coding of moving pictures and associated audio for digital storage media at up to about 1.5 mbits/s. Technical report, MPEG, 1993.

2. ITU-T Rec. H.263: Video coding for low bitrate communication. Technical report, the International Communication Union, 1996.

3. S. Martucci, I. Sodagar, T. Chiang, and Y. Zhang, "A zerotree wavelet video coder," *IEEE Trans. on Circuits and Systems for Video Tech.*, 7(1), pp. 109-118, July 1997.

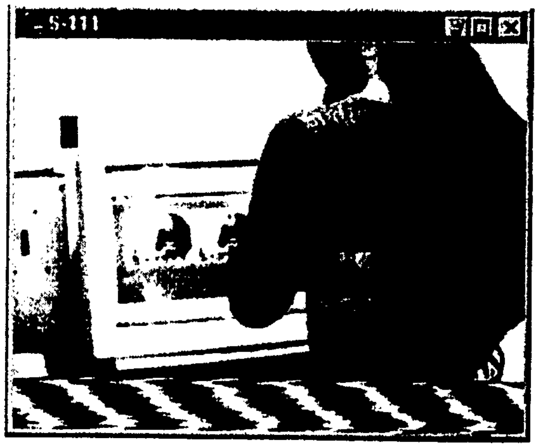
4. C. Lin, B. Zhang and Y. F. Zheng, "Packed computation of integer wavelet transform constructed by lifting scheme," *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Phoenix, AZ, pp. 3177-3180, March 15-19, 1999.

5. F. Keinert, "Biorthogonal wavelets for fast matrix computations," *1991 Mathematics Subject Classification*. 65F30, Secondary 42C15.

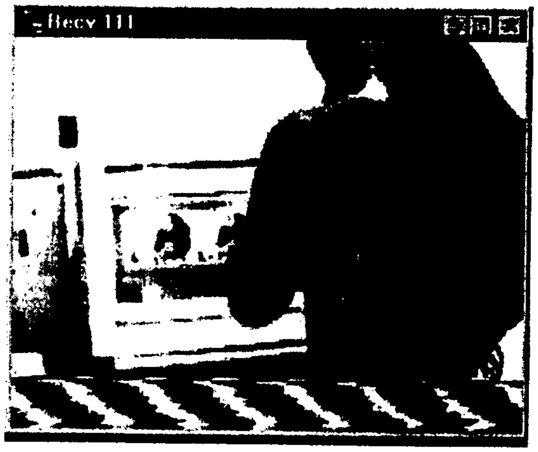
6. D. L. Mills, "Internet time synchronization: The network time protocol," *IEEE Trans Comm.*, 39(10), pp. 1482-1493, October 1991.

7. ITU-T Rec. H.225.0, Media stream packetization and synchronization for visual telephone systems on non-guaranteed quality of service LANs. Technical report, the International Communication Union, 1996.

8. ITU-T Rec. H.322, H.323: Videoconferencing standards for packet switching networks. Technical report, the International Communication Union, 1995 and 1996.



(a) Original combined frame at server



(b) Reconstructed frame at client

Figure 3.5 Comparison between display windows at server and client

TASK 17

Information Fusion and Modeling of 3D Microwave Monolithic RF with Microelectromechanical Sensors

Principal Investigator:	Prof. Hoda S. Abdel-Aty-Zohdy
Institution:	Oakland University, Oakland, Michigan
Period of Performance:	September 21, 1999 to May 31, 2002
Task:	Investigate multi-strategy signal processing algorithms for different applications. Develop algorithms for on-board learning and testing. Design and implement optimum mixed-signal integration of memory, control, interface, and conditioning circuitry to replace classical circuits. Design, implement, and evaluate a novel high-speed 4-1 analog ASIC multiplexor. Develop and evaluate information fusion and fabrication techniques for multilayered, vertically interconnected designs. Investigate optimum integrated information built around VIGILANTE together with efficient interface and data converters.

The final report for this task is unavailable. It will be included within this report when available.

A Hardware Implementation Of Genetic Algorithms For Measurement Characterization

(1) Mohammad S. Sharawi, (2) James Quinlan, and Hoda S. Abdel-Aty-Zohdy

Microelectronics System Design Laboratory
Electrical and Systems Engineering
Oakland University, Rochester, MI, 48309, USA
msharawi@oakland.edu ; zohdyhsa@oakland.edu

(1) was M.S student (2) was an Undergraduate student

Abstract—Integrated circuits that sense, receive, transmit, and process signals are the eye, ears, and nose of the biotechnical field of the millennium. To process/store/analyze signals acquired from sensors, hybrid systems with flexible and adaptable Intelligent Information Processing and Perception (IIPP) are needed. Genetic Algorithms (GA) present a suitable pre-processing operation because of their good convergence to optimum solutions and characterization capabilities. Most of GAs are built via software because of their ease of editing and customization. Less attention has been given to hardware implementation of such algorithms. In this paper we present a hardware design approach of a GA for optimum measurements representation/characterization to a following IIPP stage. A multilevel verification of the GA is performed via VHDL. In particular the design of efficient universal multipliers and dividers is addressed.
A new time efficient approach for Crossover based on preassigned least error value is proposed. The Crossover scheme is called *Half Siblings and a Clone*. Such a scheme needs 10 iterations to converge the system proposed for sensor measurement characterization. This takes 7680 clock cycles, which is only 96 μsec when implemented in the 0.25 μm CMOS technology. [1]

I. Introduction

A Genetic Algorithm (GA) hardware system prototype is developed for optimum sensors-measurement fusion and characteristic weights representation [1-8]. GAs are a family of computational models inspired by evolution. Such algorithms encode a potential solution to a specific problem on a simple Chromosome-like data structure, then applies some predefined recombination operations on these structures so as to preserve critical information. Genetic Algorithms (GA) are often viewed as optimization functions that try to give an optimum/semi-optimum solution for problems with specific constraints or criteria.

GAs have been employed for solving many complex optimization problems in numerous fields. While GAs are not perfect, i.e. they do not always find the optimal point, they are very efficient in finding near-optimal solutions significantly faster than conventional point-to-point exhaustive search techniques, specially in large solution spaces.

An implementation of a Genetic Algorithm starts by the generation of a random population of solutions, usually called the *Chromosome population*. This structure of data is then evaluated, and reproduced opportunities are allocated in such a way that those chromosomes which represent a better solution to the target problem are given more chances to reproduce than those who are considered poor solutions. The optimality of a solution is typically defined with respect to the current population average or median. How close a chromosome (solution) to the optimal solution is usually found based on the minimum distance of the chromosome from the optimal solution, which is measured by the amount of error associated with the chromosome in our case.

In [1], a GA system for optimal sensor measurements was proposed. The system architecture was based on the *Binary Search* method in minimizing the error value after each iteration. The Crossover and Mutation functions were externally controlled for functionality. In this paper, we propose a new time-efficient approach that is based on a new crossover technique that we call *Half Siblings and a Clone*. Such a technique will take care of assigning fitness probabilities to the various chromosomes, and keeps the one with the least amount of error. The system keeps on iterating until the least error chromosome is found. After which the best chromosome can be used to get the best measured input value. The details of such a system and its hardware implementation and simulation via VHDL are presented. Simulation results and various illustrations are given.

II. Basic Operation

A GA uses a fitness criteria to determine the best choice of weights that should be applied to an input data in order to get the Optimum Measurement value. GAs randomly select solutions from a predefined solution space, and start applying the fitting criteria, which is followed by *Mutation* and *Crossover* to obtain the best solution according to the fitting criteria. In our proposed design, Crossover is the operation considered in finding the best fit, this is due to the fact that mutation has usually less probability of occurrence in GAs, that is why we are focusing on the Crossover operation only after the error is calculated [2]. A random population of data is initially created, which is called the *Chromosome Population* (i.e. the solution space in this case). A predefined fitness criteria is then applied on the population and on the input data measurement. Based on the results obtained, the chromosomes that survive (meet) the fitness criteria best are kept, others are

[1]This work is partially funded by a Research Grant from the Air Force Research Laboratory/Wright-Patterson Air Force Base/Sensors Directorate, PRDA No. F33615-96-2-1945, 2000-2002

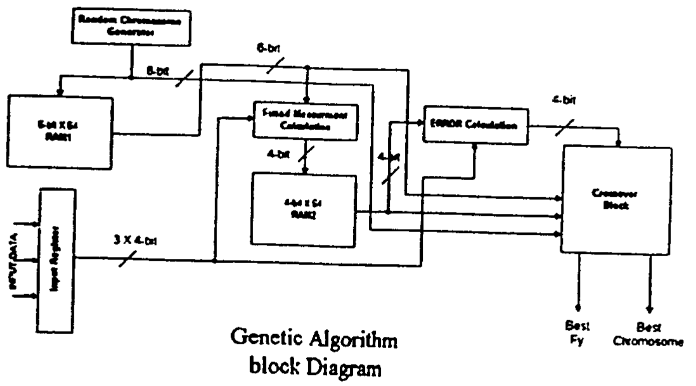


Fig. 1. Block Diagram of proposed GA.

replaced (or Crossovered). The algorithm goes into a number of iterations to get the optimum chromosome, which is then applied to the incoming data to get the optimum value for that specific input. A block diagram of the proposed implementation of the GA is shown in Fig.1. The chromosome population is created by a Random Chromosome Generator (RCG), and stored in a Random Access Memory (RAM). The GA starts its iterations by calculating the Fused Measurement based on the input data, and the random chromosomes. The results are stored, and later used in the calculation of the Error value. These two steps resemble the fitness criteria. The calculated error signal is then used to determine if the chromosome will survive this iteration, be replaced, or crossovered with another generated chromosome. Based on the chromosome's fitness, it is either kept or dropped. The chromosome is assigned a probability indicating its closeness to the optimal fit, then it is kept for further Crossover with less fit chromosomes to generate fitter chromosomes (called offsprings). A flowchart describing the algorithm flow is shown in Fig.2.

III. Hardware Description

In this section, we will present the detailed hardware description of each of the building blocks of the GA proposed in Fig.1. The description is based on our VHDL hardware design.

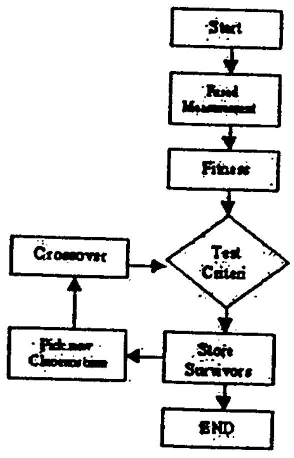


Fig. 2. Flowchart for the proposed GA Hardware.

A. Random Generator

The Random Chromosome Generator is concerned with generating the population needed for the GA. Each chromosome is 6-bits long, and represents 3/ 2-bit *Genes*. Each gene is used with its respective input measurement to calculate the fused measurement needed for the fitting criteria. To generate these chromosomes, a *Linear Feedback Shift Register* was used [1].

B. Fused Measurement Calculation

The Fused Measurement (FM) block has two inputs; the input measurement which comes from the input register that stores 3 different measurements for the input quantity, and a chromosome from the generated population. The fused measurement is then calculated using the equation:

$$Y_F = \frac{Y_1 \times W_1 + Y_2 \times W_2 + Y_3 \times W_3}{W_1 + W_2 + W_3} \qquad (1)$$

The Fused Measurement calculated, will then be used to find the error of the input signal with respect to this calculated value. At the end of this process, the 4-most significant bits of the 64 outputs are stored in RAM2. The 64 chromosomes are still in RAM1.

C. Error Calculation

The *Error Calculation* is the difference between the fused measurement and the actual input value. The calculation is done based on:

$$error = | Y_1 - Y_F | + | Y_2 - Y_F | + | Y_3 - Y_F | \qquad (2)$$

Once the fitting process takes place, it finds the deviation from the fused measurement, which will aid in the chromosome selection process.

After a single Error value is calculated, its 4-Most Significant Bits (MSB) are applied directly to the *Crossover block* [1]. The Crossover block is discussed next.

D. Crossover Block

Traditional genetic algorithms employ a Crossover scheme in which the probability of a chromosome being selected for Crossover is proportional to its fitness. This scheme requires that all the chromosomes be compared to each other, a process that would take much time in hardware.

To eliminate the time-cost, this system incorporates a form of Crossover called *"Half Siblings and a Clone"*. This Crossover scheme uses the chromosome with highest fitness as the first parent, and all of the chromosomes in sequence as possible second parent. It requires that the Error of each chromosome be compared to a stored Least Error value. This value is initially the highest possible Error value and loaded into a capacitor. If a chromosome's Error is less than this value, then the capacitor's voltage lowers to this new value, and the chromosome is loaded into parent Slot 1. Traditional genetic algorithms often employ mutation to increase function space exploration. This system incorporates a form of mutation into its Crossover scheme.

Each chromosome has an additional bit called a Fit Bit. A Defining Fitness Voltage parameter is pre-defined in the chip design, or left for user input. When the chromosomes have their Error compared to the Least Error voltage, they also have it compared the defining Fitness Voltage. If the Error is less than this parameter, then the chromosome's fit bit is set to "1", otherwise the fit bit is a "0". The importance of this fit bit is that when a chromosome is to be loaded in parent Slot 2, its fit bit is checked. If the fit bit is a "0", then a randomly generated chromosome is loaded. This scheme incorporates data variety in the initial iterations, but doesn't inhibit convergence in the final iterations. The given simulations used a Defining Error Voltage of one-fourth the Maximum Error Voltage.

In Fig.3,4, the proposed Crossover scheme is illustrated. In the first, the fitness based on the calculated error value is assigned to the chromosome via the digital current detectors and the diodes. These two elements can be resembled by a voltage comparator for simplicity. Where the capacitor that holds the error voltage is resembled by a digital accumulator when implemented in VHDL. This accumulator can be incremented and decremented based on the error voltage feeding the comparator. If the voltage of the error is less than that of the accumulator then that chromosome is loaded into Slot 1, the accumulator decrements to this new lowest level, and the comparisons continue. If the voltage of the error is not lower, then the comparisons simply continue. Such a comparison is done for all the population. By the end of this cycle, we will have the chromosome with the least error being sorted and put in Slot 1.

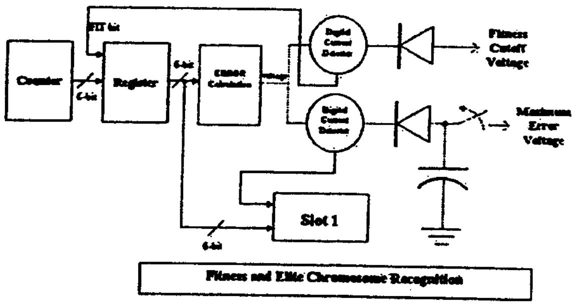


Fig. 3. Crossover block I.

In Fig.4, the Crossover operation is demonstrated. With the Fit Bits already assigned and the chromosome with the least error loaded into Slot 1, we can produce a new generation with just two clock cycles per new chromosome. To achieve this speed, we limit our crossover to crossing single digit values with the same gene and digit place value. When coupled with a fifty-percent crossover chance per digit, we achieve a crossover scheme that be realized with simple combinational logic, using a random chromosome as our crossover determining value.

During the first clock cycle, the chromosome is ANDed with its Fit bit, while a random chromosome is ANDed with the negated Fit bit. The results of each are ORed together and loaded into Slot 2. During the next clock cycle, chromosome in Slot 1 is ANDed with a new random chromosome while the chromosome of Slot 2 is ANDed with the negation of the random chromosome. The results of each are ORed together and ready to be loaded back into the register.

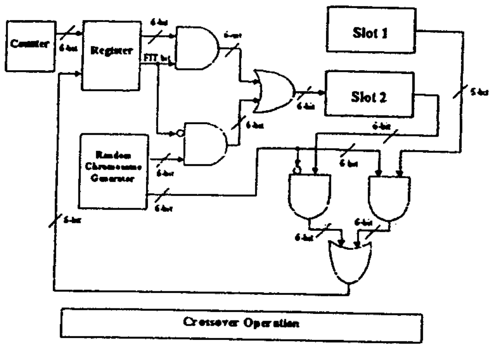


Fig. 4. Crossover block II.

IV. Results and Discussion

In this prototype a static GA approach has been used to prove the feasibility of FPGA system implementation. A static input with relatively small measurement values, and a predefined solution space has been used. Simulation results are illustrated in Fig.5, Fig.6, and Fig.7. In Fig.5, the convergence of the chromosome is illustrated for a random set of input data (6-bits total). The GA keeps on iterating, until the genes with the least error are generated via Crossover and selection processes. It is shown that the GA reaches its optimum chromosome value after 8 iterations (cycles). The figure shows the fitness of the random chromosome relative to the pre-calculated best chromosome for the applied measurements with zero error. The standard deviation of the relative iteration is also demonstrated.

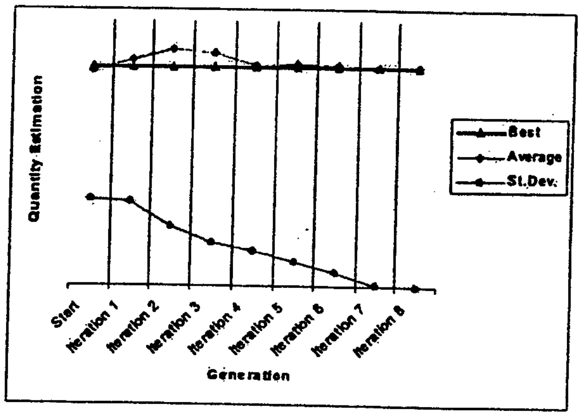


Fig. 5. Convergence of a random 2-bit gene based on 6-bit input.

In Fig.6, another set of input values are chosen, and the GA starts its iterations searching for the optimum chromosome that minimizes the error calculated value of the fitting criteria. This time it takes the algorithm 9 iterations to reach the optimum solution with minimum error. The figure shows the convergence process relative to the best fit. The Fused measurement, the optimum chromosome that survived the Crossover and fitting functions, and the optimum calculated measurement can be obtained from the system after the final iteration is over. The optimum chromosome, and measurement are then further used by subsequent stages in the system. In Fig.7, the convergence of all the chromosome population to the best fit is illustrated as the number of iterations increases. For such a system, the total clock cycles needed can be calculated as indicated in Table I.

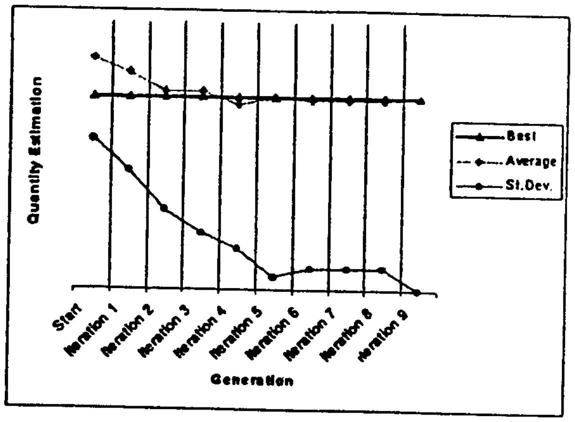


Fig. 6. Convergence of another set of random inputs.

Comparison with other GA schemes like *Table Lookup* and *Polynomial fit* is not directly applicable, since our design resembles a prototype scheme, and uses a relatively small solution space compared with the *Polynomial fit* used in [3]. Not to mention that most of the approaches are not implemented in hardware. This new approach will yield a 20% savings in the number of CLBs (HW components), and 15% in convergence time with respect to previous work [1]. The presented prototype has a limited data width of 6-bits, which can be expanded for a practical application.

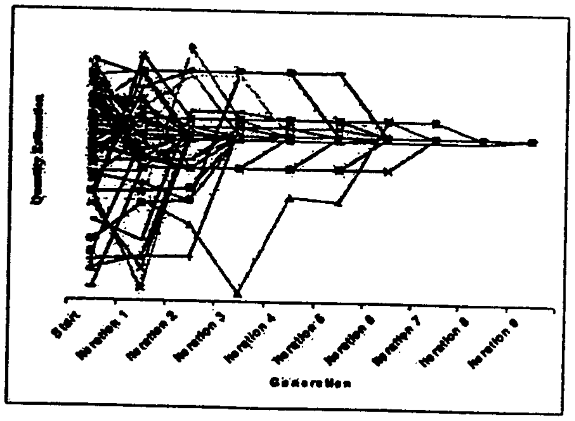


Fig. 7. Chromosome population convergence.

V. Conclusions

A GA optimization system has been implemented using VHDL. Experiments have been conducted to test its performance. Results show that the system is effective, small and fast. Also, the error convergence improves as the iteration goes on. Based on the initial generated population, the total system's cycle of operation has been determined. It takes at most 10 iterations to converge to the optimum chromosome (solution) with 7680 clock cycles, 96 μsec when implemented in 0.25 μm CMOS technology. It is 20% HW efficient and 15% time efficient compared to [1]. The presented GA system provides sensor fused measurement and representative weights (chromosomes) to an IIPP system. Thus, the presented GA system enables the necessary hardware for direct interface between sensors and neural networks in real-time, practical and effective applications.

TABLE I
Basic Timing requirements of the proposed GA.

Operation	Min Number of CLK cycles
RCG and Filling RAM1	64
FM computation, and storing	64×4
Error Calculation	3 CLKs per Chromosome
Fitness and Elite	2 CLKs per Chromosome
Crossover	2 CLKs per Chromosome
One Iteration Total	768
Total time	$(768 \times n) + (P_D \times n)^*$

* P_D is the propagation delay of the combinational logic.
n is the number of algorithm iterations.

References

[1] Mohammad S. Sharawi, H. S. Abdel-Aty-Zohdy and R. L. Ewing. "Optimal-Weights Sensors-Measurement Fusion using Genetic Algorithms". *IEEE Proceedings*, Mid-West Simposiom on Circuits and Systems, pp. 537-542, August 2001.
[2] Cem Ergün and Kadri Hacioglu. "Multiuser Detection Using a Genetic algorithm in CDMA Communications Systems", *IEEE Transaction on Communications*, VOL.48, NO.8, pp. 1373-1383, August 2000.
[3] James W. Hauser and Carla N. Purdy. "Sensor Data Processing Using Genetic Algorithms". *IEEE Proceedings*, Mid-West Simposiom on Circuits and Systems, 2000.
[4] Olivier Allegre and Hoda Abdel-Aty-Zohdy, "Research Assistantship in a Sensor Optimization Project based on Genetic Algorithms", *Project Report*, Oakland University, Summer 2000.
[5] Mohammad Zohdy, Djamel Bouchaffra and James Quinlan. "Optimal Mapping from Chromosome Space to Feature Space For Solving Sequentional Pattern Recognition Problems", *IEEE Proceedings*, Mid-West Simposiom on Circuits and Systems, pp. 520-525, August 2001.
[6] Eckart Zitzler and Jürgen Teich and Shuvra S. Bhattacharyya. "Evolutionary Algorithms for the Synthesis of Embedded Software", *IEEE Transactionas on Very Large Scale Integration Systems*, VOL.8, NO.4, pp. 452-455, August 2000.
[7] H. S. Abdel-Aty-Zohdy, R. L. Ewing, M. C. Mah, and L. Liou, "Integrated Circuits and Systems for Solving Bio-Chemical Sensor Development and Detection Problems," *Fifth Joint Conference on Standoff Detection for Chemical and Biological Defense, (5JCSD), Proceedings* pp.xx-xx+16, September 2001, Williamsburg, Virginia.
[8] H. S. Abdel-Aty-Zohdy, *"Artificial Neural Network Electronic Nose For Volatile Organic Compounds," IEEE Computer Society Press for the Eighth Great Lakes Symposium on VLSI*, pp. 122 - 125, Lafayette, Louisiana, February 1998.

TASK 18

Development of 3D Vertically Interconnected Microwave Integrated Circuit (3D IC)

Principal Investigator: Prof. Altan Ferendeci

Institution: University of Cincinnati, Cincinnati, Ohio

Period of Performance: August 20, 1999 to June 10, 2002

Task: The contractor shall investigate different dielectric materials for developing an electronic nose to detect an environmental gas component specified by AFRL. The investigative initial study will be concentrated on literature search of existing electronic nose systems with emphasis on coming up with a new electronic nose that is based on detection of the gases using planar microwave resonators processed by MEMS technology.

3D Multilayer Microwave Integrated Circuit (3DIC)

Quarterly Report (Covering April – June. 00)

1. Status of Each team members including progress and technical discussions
   - UC

Two major components were investigated during this period: the antenna and antenna feed circuitry. Because of the limited bandwidth and efficiency of the patch and cavity backed antennas reported in earlier reports, slotted spiral antenna was chosen as a possible radiating element for the 3D-IC module. Since the antenna is a symmetric structure, it requires a balanced feed mechanism. To provide transformation from stripline to a balanced structure, various balun configurations were investigated.

WIDE-BANDWIDTH BALUN

Various balun structures were investigated. The requirement for these were that the geometry was structurally compatible with the available 3D processing technologies. Among the possible balun, the following structure developed by UCLA [1,2] was modified for our purposes.

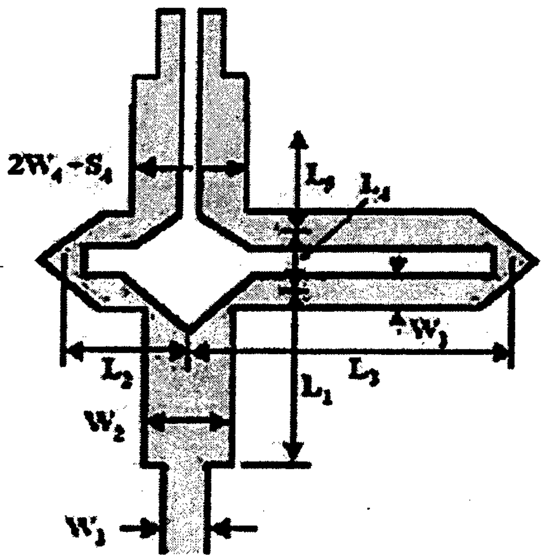


Figure 1. Balun structure

Figure 1 shows the layout of the balun. Since the original design was for microstriplines, the structure was modified to a stripline version. Even though the right hand line introduces the needed 180^o phase difference, the two $\lambda/4$ coupled line has an all pass characteristics.

The structure was simulated as a microwave circuit using HP-Libra. Coupled stripline were used with image impedance of 50 Ω.. The simulated results are shown in Figures 2 and 3.

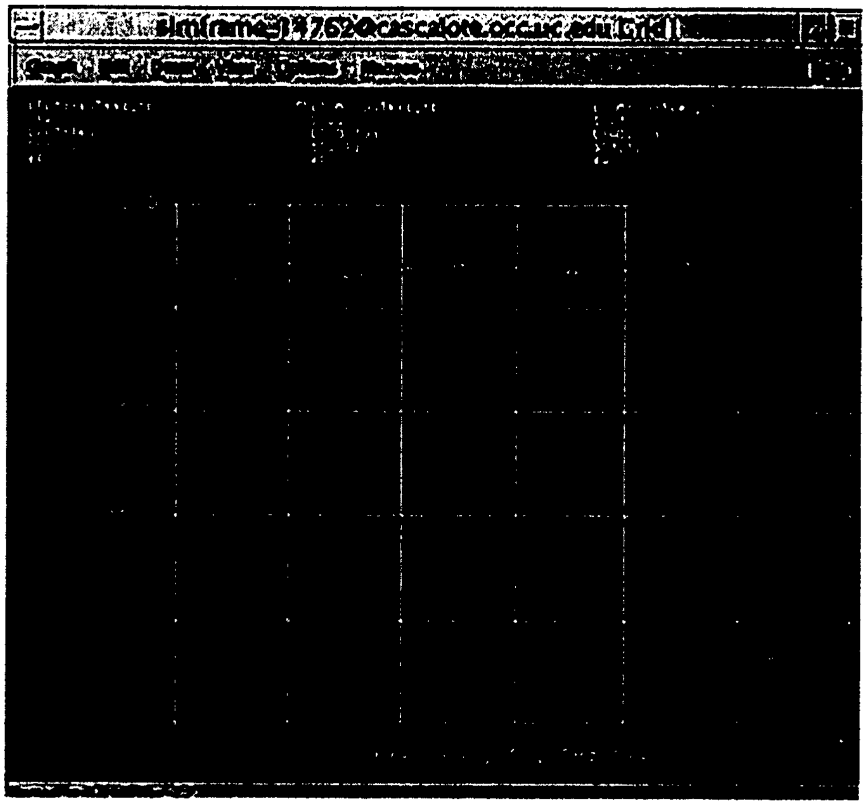


Figure 2. Magnitude of S_{12} S_{13} and S_{11} as a function of frequency

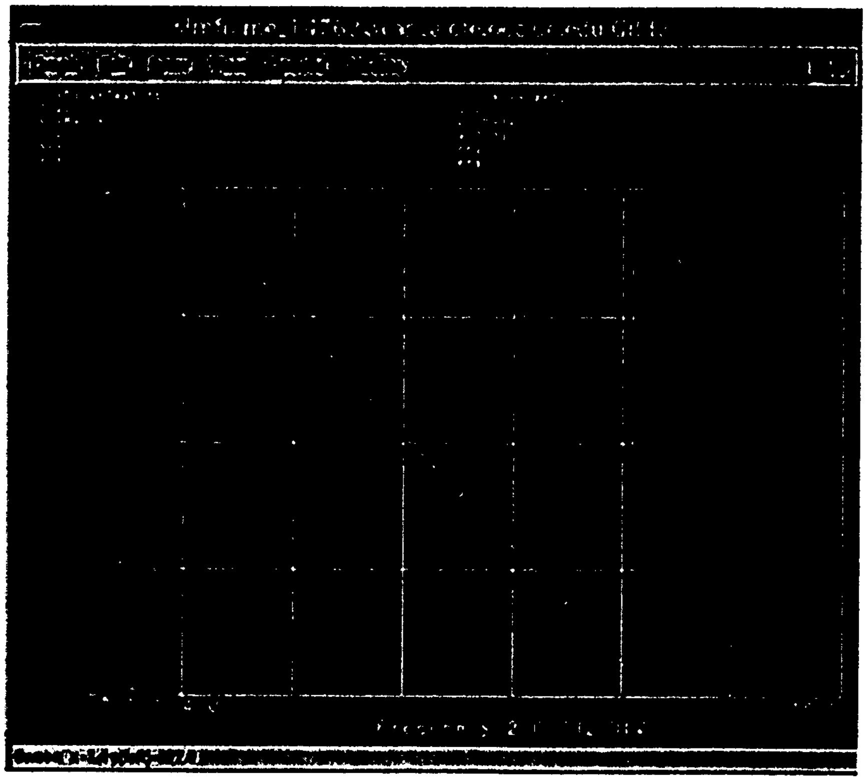


Figure 3. Phase between S_{12} and S_{13} as a function of frequency.

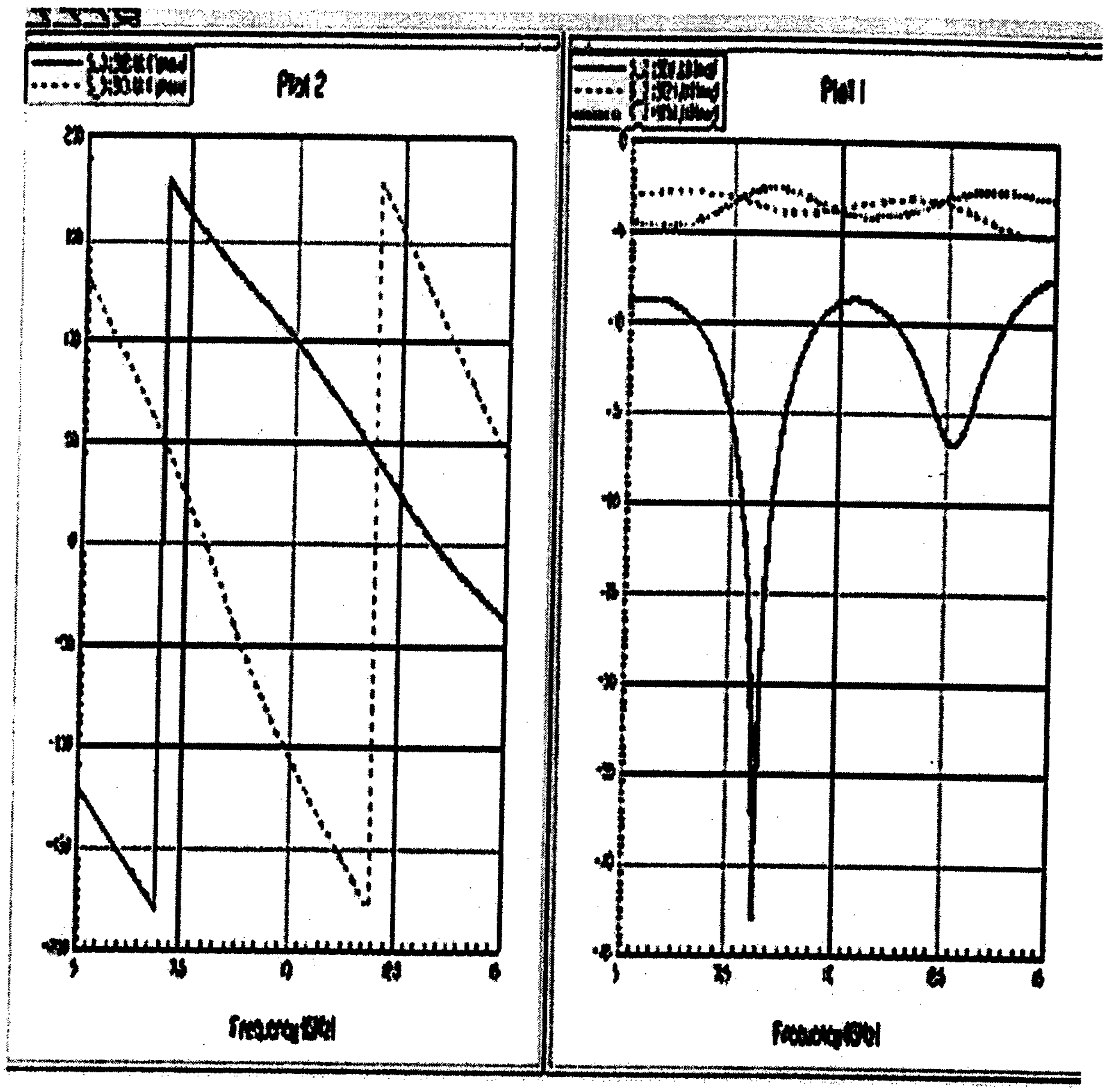


Figure 4. Phase between S_{12} and S_{13} and magnitude of S_{11}, S_{12}, S_{13} and S11 obtained using HFFS.

The same structure was also simulated using Ansoft HFFS and the results are shown in Figure 4. As can be seen from both of these simulations that there is considerable uniform power division between the two ports over the frequency range of 5-15 GHz. The total dielectric thickness was 44 μm.

ANTENNA CONFIGURATIONS

Three antenna configurations were considered.

a) patch antenna
b) cavity backed slot antenna
c) slotted spiral antenna

The performance of the patch and cavity backed antenna were presented in our previous reports. In this report emphasis is placed on the wide bandwidth slotted spiral antenna. Since the dielectric layer between the backing ground plane and the antenna is still thin, a slot type antenna was chosen over the a strip type spiral.

An antenna was designed to operate between 5-15 GHz. The antenna was fed with a simple stripline which is divided into two paths with an additional $\lambda/2$ line in one of the arms. The dielectric thickness was chosen to be 44 μm. The structure was simulated using Ansoft HFFS. The structure of the antenna is shown in Figure 4.

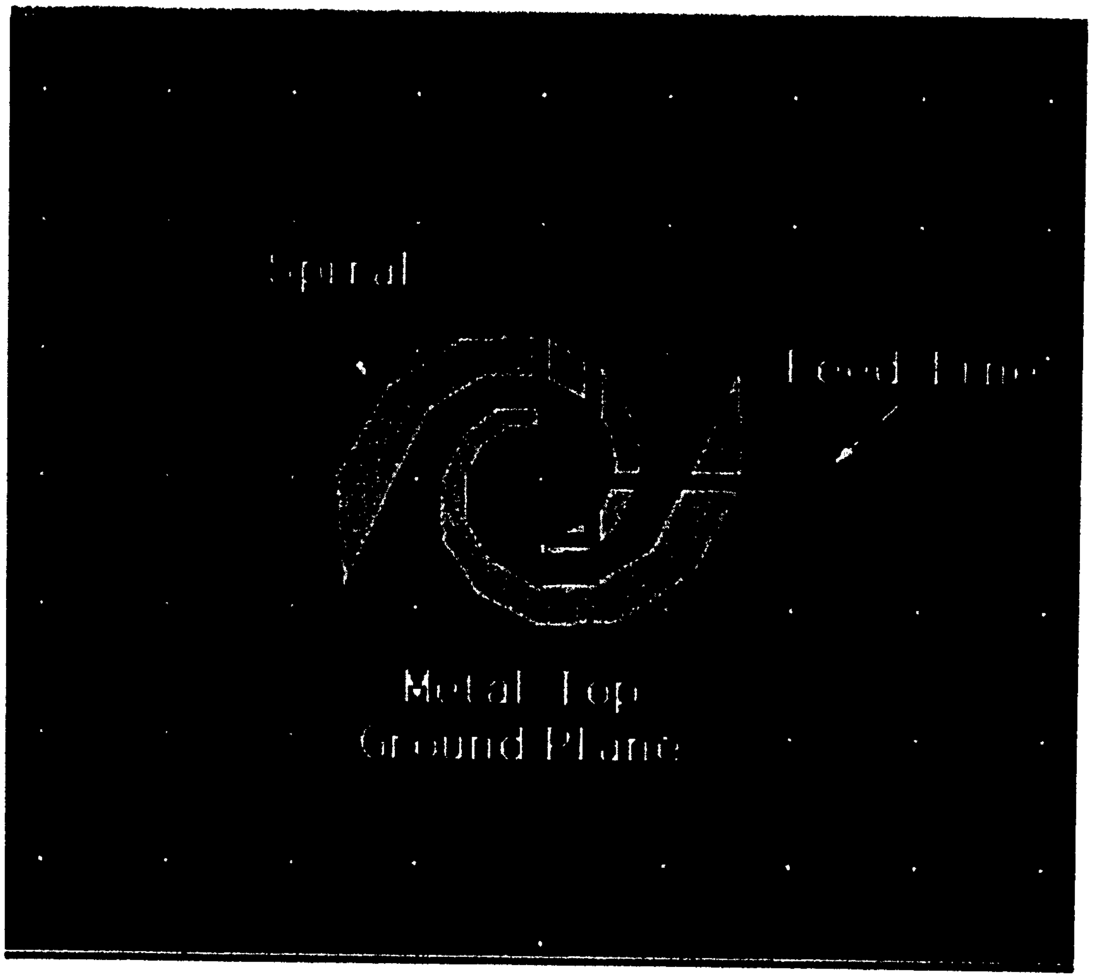


Figure. 4 Top view of the spiral antenna. The feed line is shown in red color.

The E-field at the top surface and the directivity at two different frequencies are shown in Figure 5. A can be seen from the directivity plots, the antenna has a wide frequency response.

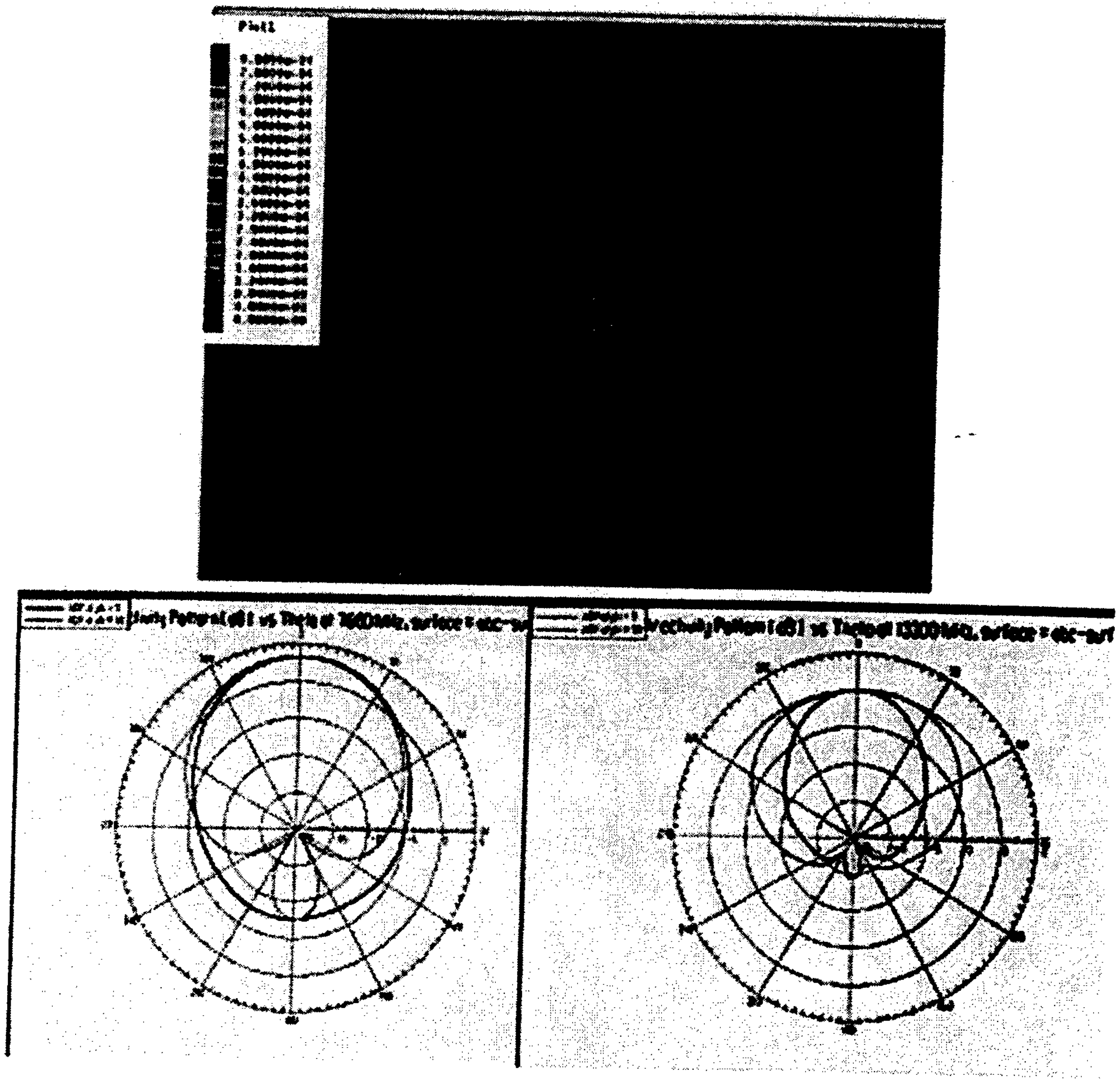


Figure 6. (a) E field distribution on the top surface and directivity at (b) 7.66 and (c)13.3 GHz.

REFRENCES

[1] Y. Qian and T. Itoh, "A broadband uniplanar microstrip-to-CPS transition," 1997 Asia-Pacific Microwave Conf. (APMC'97) Dig., pp. 609-612, Dec. 1997.

[2] N. Kaneda, Y. Qian and T. Itoh, "A novel Yagi-Uda dipole array fed by a microstrip-to-CPS transition," 1998 Asia-Pacific Microwave Conf. (APMC'98) Dig., pp. 1413-1416, Dec. 1998.

PUBLICATIONS (during this period).

"Interlayer MEMS RF Switch for 3D MMICs," Yu Albert Wang, Qinghua Kang, Bosui Liu, Altan M. Ferendeci, and Misoon Mah, International MTT-S Microwave Conference 00, Boston, MA (June 12-15, 2000)

Vertically Interconnected 3D Module for Conformal Phased Array Antennas Bosui Liu, Yu Albert Wang, George Kang, and Altan M. Ferendeci and Misoon Mah, IEEE International Conference on Phased Array Systems and Technology, Dana Point CA, (May 21-25, 2000)

"Vertically Interconnected Multilayer Microwave Circuits", Altan M. Ferendeci, Bosui Liu, George Kang and Misoon Mah, 1999 Government Microcircuit Applications Conference (GOMAC 2000), Anaheim, CA (22-24 March 2000).

Progress Report
July-1, September-30, 2000
Submitted by University of Cincinnati

During the months of July-September 2000, in addition to the continuing work on antenna and MEMS switch , the following two major areas were investigated.

These are a) processed development and b) power amplifier.

Process Development
One of the essential component of 3D-IC realization is the successful growth of vertical posts and isolation walls. This involves thick layers of metal deposition thickness ranging from 10-50 μm with diameters ranging from 10-50 μm. Two types of post growth are necessary: a) posts over ground planes and b) posts over circuit planes.

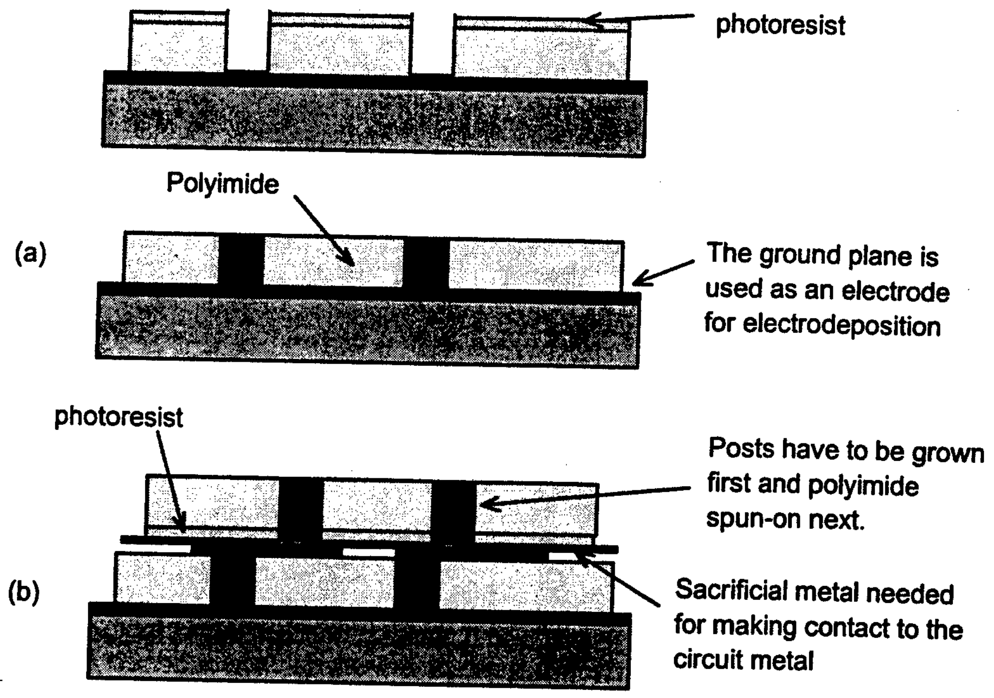

photoresist
Polyimide
(a)
The ground plane is used as an electrode for electrodeposition
photoresist
Posts have to be grown first and polyimide spun-on next.
(b)
Sacrificial metal needed for making contact to the circuit metal

Figure 1. Post growth processes.

a) Processes involved for posts over a ground plane are shown in Figure 1a above. Here polyimide was first deposited over the ground plane followed by a thin metal layer which was used as a mask. Photoresist was spun over this metal layer. Photo resist was exposed and opening in the metal mask was etched away. Using RIE, post opening and trenches for the isolation walls were etched away until the lower ground plane was reached. Using the ground plane as one of the electrodes, gold was deposited through the openings. Finally, the mask metal was removed and the surface was planarized.

b) When a circuit element is to be connected to an upper layer through a post, in many cases the circuit elements are have no physical connection to each other. Since this circuit layer is to be covered with a top dielectric layer, there are no ways to make direct electrical contact to each circuit element for metal deposition. To solve this problem, the following procedure was being investigated. This is shown in Figure 1b.
   1) circuit plane was first processed,
   2) a thin metal layer as a sacrificial layer was deposited over the whole circuit plane
   3) photoresist was used to open the location of the posts and trenches
   4) using sacrificial metal layer as one of the electrodes, posts and isolation walls were grown through these openings.
   5) The sacrificial metal layer was removed
   6) Polyimide was spun over the circuit plane
   7) The top layer was planarized.

   Steps 1-5 were already successfully demonstrated. One of the major concerns with this technique was the maintaining the posts and walls during the polyimide spinning. Various post and wall geometry were used to test the strength of the grown metals.

Power Amplifier

In order to process a 3D power amplifier circuit, a hybrid amplifier approach was taken as shown in Figure 2.

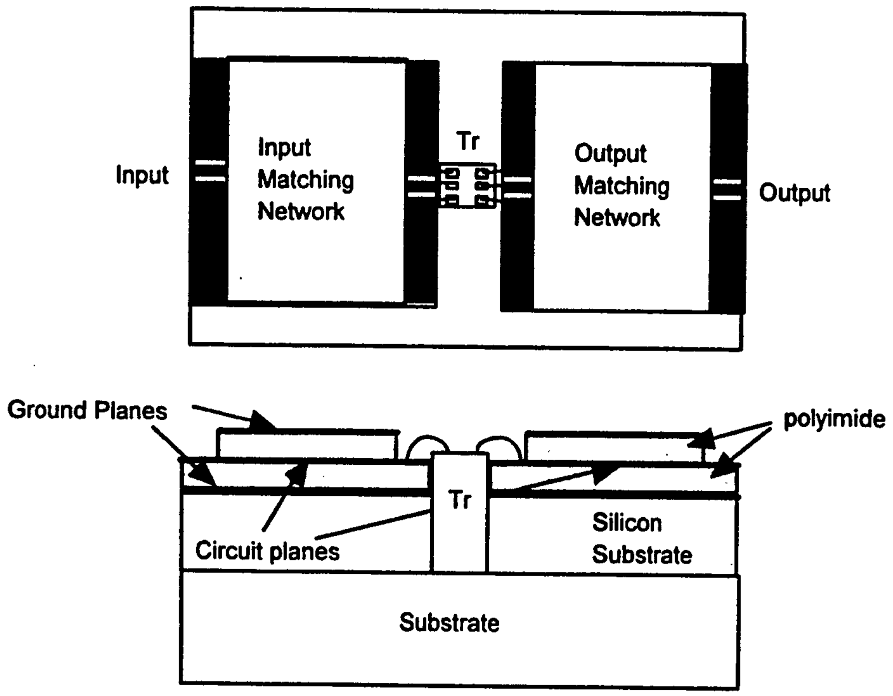


Figure 2. Hybrid 3d power amplifier

A silicon wafer was used as a substrate. An AMCOM chip transistor was epoxied to the substrate. A second Si wafer was used to build the matching networks. A ground plane was first deposited followed by polyimide layer. A circuit plane containing the input and output matching networks was then deposited and processed. Next, the top polyimide and the top ground planes were deposited. Using RIE, the top layers of polyimide were etched away to expose the input and output ports of the matching networks. The matching networks were then diced and epoxied to the substrate. The transistor was wire bonded to the matching networks. The input and output ports were used to test the amplifier.

The power amplifier circuit was designed using the load pull data supplied by AMCOM. The matching networks specifically designed using the load pull data. Simulated data shows that the amplifier gain is 6.5 dB with bandwidth of 13% at 10 GHz.

The masks were already processed and various depositions were taking place to implement the power amplifier circuitry. Once the amplifier is successfully tested, next step is to extend this technique to multiple amplifying stages to increase the gain and power output.

Note: The paper

"Monolithically Processed Vertically Interconnected 3D Phased Array Antenna Module" Paper 104, 51st National Aerospace and Electronics Conference, NAECON 2000, Tipp City, OH (Oct 2000)

won the **First Place Award** for Best Paper in the Da Vinci Predictive Paper Award Category at the **2000 National Aerospace and Electronics Conference**

Progress Report
Oct-1, Dec-31, 2000
Submitted by University of Cincinnati

During the months of July-September 2000, in addition to the continuing work on antenna and MEMS switch , a wide-bandwidth balun was investigated.

Wide-bandwitdh 3D balun

Considerable research and development have been conducted for the development of three-dimensional Multilayer Monolithic Microwave Integrated Circuits (3DIC) [1]. A basic X-band 3D IC/MEMS microwave transmit module has been developed. [2]. A slotted spiral antenna was currently developed as a radiator of the affordable conformal array T/R module due to its intrinsic wide bandwidth radiation characteristics.

A balanced transmission line was required to feed the spiral antenna through two vias that act as two parallel wires. At the feed position, electric fields should be of a equal amplitude and 180° out of phase. Typically, unbalanced stripline was the most natural form of transmission lines and can be easily integrated with the 3D multilayer monolithic microwave circuits. A balun was thus necessary to realize an efficient transition between striplines and balanced feeding vias. A type of uniplanar microstrip to CPS transition based on the concept of mode conversion was resulted in 3dB insertion loss and bandwidth of 49% for a balanced back-to-back configuration [3]. Further improvement of 68% bandwidth was achieved with the similar insertion loss by employing a symmetrically optimized T-junction for signal dividing/combining, and by using optimal miters for 90° bends [4]. Both of these designs incorporated conventional microstrip lines, and were fabricated on a RT/Duriod substrate with a dielectric constant around 10.5. The topside of the uniplanar microstrip line balun was exposed to air, which made them unsuitable for 3D multilayer microwave integrated circuitry. Furthermore, undesired surface waves caused by circuit discontinuity were unavoidable, and thus, microwave-absorbing material was used during the test [3].

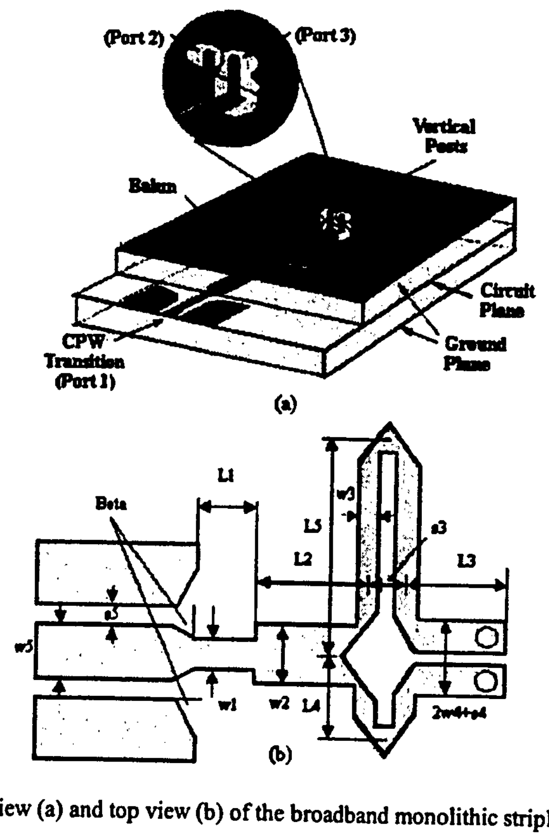


Fig. 1. Perspective view (a) and top view (b) of the broadband monolithic stripline balun.

A stripline balun that was completely compatible with a 3D multilayer monolithic microwave circuitry was developed. The balun was 65 μm thick and was monolithically processed on an 2" Silicon wafer. Multiple layers of polyimide were used as the dielectric material with a low dielectric constant. This resulted in wider line widths for the given characteristic impedance. The design was simulated and optimized using HFFS. For the fabricated back-to-back stripline balun configuration, an insertion loss of less than 1.2 dB was achieved over 32% bandwidth and of 0.95 dB around the designed frequency of 13 GHz. The input return loss was better than 10dB within the above bandwidth.

II. DESIGN GUIDELINES

Fig.1 shows the configuration of the 3D balun for operation at 13 GHz. The stripline at the input port as well as the two branches were all designed to be 60 Ω. The output impedance of the balun was optimized for 120Ω. A quarter wavelength transformer of 42.4Ω together with a T-junction connects the input stripline with the two output paths.

The function of the balun was to transfer the input power on an unbalanced line, the stripline in the present case, to a balanced transmission line at the output. The two output lines form a pair of coupled striplines due to their close spacing. Since the difference between two branches was designed to be a half a wavelength, the propagation mode in the output coupled stripline was dominantly the odd mode. The configuration of those two stripline branches was symmetric, which optimized the in-band return loss [4]. In order to reduce the undesired reflections, the T-junction and all the 90-degree stripline bends were compensated by trimming off part of conductor [5].

The stripline at the input port of the 3D balun can be directly connected to other multilayer integrated circuits, such as phase shifters, filters and power amplifiers. It was completely compatible with the 3DIC module. However, in order to access the balun for experimental testing and verification, a CPW with bottom ground plane (CPWG) to stripline transition was added to the input port as shown in Fig.1 and 2, extending the work of Houdard et al. [6] at W-band frequencies. The choice of the coupling constants for the two phase-arms were optimized using HFFS.

III. EXPERIMENTAL RESULTS

Based on the above design guideline, a prototype balanced back-to-back stripline balun and a three-port version were designed, processed and tested for operation at 13 GHz.

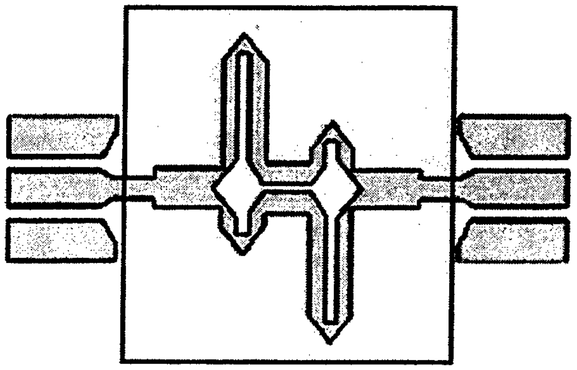


Fig.2. Balanced back-to-back stripline balun configuration

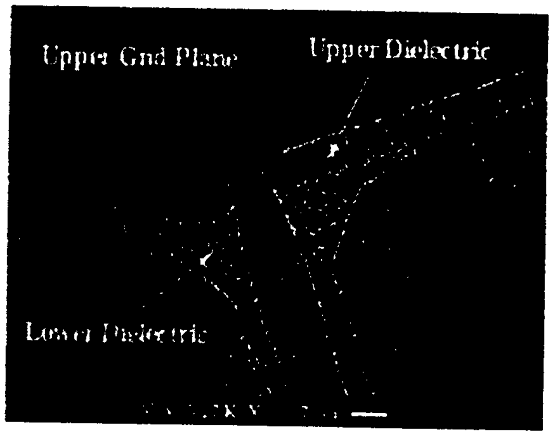


Fig.3. SEM image of the fabricated CPWG to stripline transition

The balanced back-to-back stripline balun configuration was shown in Fig.2. Two identical stripline baluns were placed on opposite sides along the horizontal centerline. An SEM image of the CPWG to stripline transition of the fabricated balun was shown in Fig.3. The dielectric layer and part of the top ground plane covering the CPWG pads were removed for the access of the testing probes. The CPWG to stripline transition was simulated and optimized using Finite Element Method (FEM), and its insertion loss was less than 0.02 dB over the entire operation band.

Polyimide (PI2611, DuPont) was chosen as the dielectric material with a dielectric constant of 3.3 in Ku-band. The low dielectric constant of the PI2611 yields a wide stripline width for the ease of fabrication. All conductors were sputtered Cr/Cu/Cr with a thickness of at least 1.2 μm, corresponding to 2 skin depths of copper at 13 GHz.

A. Processing Procedure

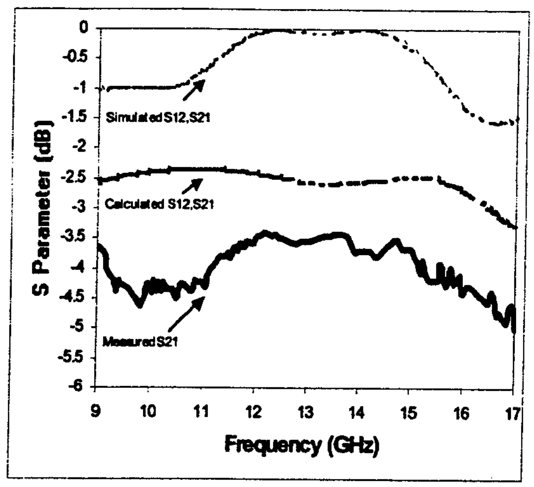
The following processing steps were used to fabricate the monolithic stripline balun:

- Lower ground plane metal was deposited over a 5 cm Silicon wafer. The thickness of Cr/Cu/Cr layers were 0.05 μm, 1.2 μm and 0.05 μm respectively. Cr served as the adhesion layer.
- For the bottom dielectric layer, 4 layers of polyimide (PI2611, DuPont) was spin-coated, and then hard cured. Total thickness of the polyimide layer was 31 μm.
- For the circuit plane, a second Cr/Cu/Cr was sputtered and then patterned by lithography and wet etching to form the necessary circuit elements.
- A second 4 layers of polyimide (PI2611, DuPont) was spin-coated over the circuit plane, and then hard cured. Total thickness of the upper polyimide layer was also 31 μm.
- As an upper ground plane of Cr/Cu/Cr were then sputtered. An Aluminum layer of 0.1 μm thick was deposited as a mask for reactive ion etching (RIE).
- RIE was performed to remove polyimide layer over the CPWG pads and at the same time open holes for vias. The total etched polyimide layer was 31 μm.
- Finally, via posts were electroplated through via holes.

B. Simulation and measurement results

Fig.4. shows the simulation and experimental results of the prototype balanced back-to-back balun configuration shown in Fig.2. The structural simulations were done using Ansoft HFSS and the *S* parameter measurements were performed using an HP8510C Network Analyzer. In addition, the circuit simulations were done using HP Libra.

The overall agreement between the experimental and theoretical results were highly satisfactory considering the slight differences between the theoretically analyzed structure and the experimentally characterized one. These differences were due to the slightly narrower lines that occurred during the processing.



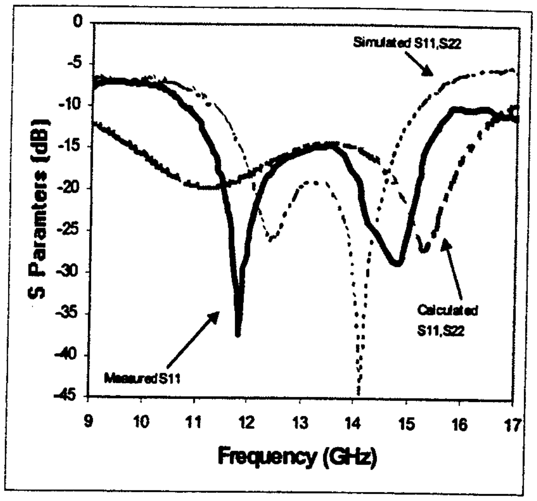


Fig.4. Calculated, simulated and measured *S*-parameters of the balanced back-to-back stripline balun.

Specifically, for the HFFS simulations an infinitely thin perfect conductor was used for the stripline metal. On the other hand, the 1.2 μm thick copper layer used in the fabrication of the stripline balun had a finite conductivity and larger than 0.85 dB attenuation per guided wavelength. Furthermore, since the characteristic impedance of the input stripline and CPWG were both 60 Ω, a mismatch occurred during the measurement when the testing probes were calibrated at 50Ω. This mismatch resulted in an extra 1.66 dB insertion loss when a two-port measurement were performed. In the HP Libra simulations, finite conductivity and finite thickness of the copper were taken into account.

As shown in Fig.4, the measured insertion loss of the balanced back-to-back stripline balun was better than 4 dB in the frequency range 11.12-15.26 GHz and translates into 1.2 dB insertion loss over a 32% bandwidth at a center frequency of 13.19 GHz. At the design frequency of 13 GHz, the insertion loss was only 0.95 dB . And the VSWR was less than 2 from 10.8 GHz to 15.66 GHz.

During the measurement, no microwave absorber was placed at the edges of the substrate to absorb the parallel plate mode waves launched by the stripline discontinuities. The impedance mismatch can easily be eliminated by changing the characteristic impedance of the input stripline and CPWG to 50Ω.

C. 3-Port measurement

Together with the prototype balanced back-to-back balun, a three port single balun of the same design was also fabricated with vias plated through the top dielectric layer as shown in Fig.1. A rigorous technique for measuring the scatter matrix of a multiport device with a 2-port network analyzer measurements was available [7]. A third probe was added into thee probe station in order to determine the magnitude and phase difference between two output ports. The 3-port measurement results of the fabricated balun will also be presented.

IV. CONCLUSION

A 3D broadband monolithic stripline balun has been designed and characterized both experimentally and theoretically. The balun was simple, compact and fully compatible with the three-dimensional multilayer microwave transmitter module design. The performance of the balun was evaluated numerically and experimentally from 9-17 GHz and both results were in close agreements. For the balanced back-to-back balun configuration, an insertion loss of less than 1.2 dB was achieved over 32% bandwidth at the design frequency of 13 GHz. The VSWR was less than 2 within the entire bandwidth.

REFERENCES

[1] Altan M. Ferendeci, Bosui Liu, Ho Huang, Missoon Mah, and Lee Liou, "3D Multilayer Monolithic Microwave (M3) passive transmitter module", *IEEE MTT-S Int. Microwave Symp. Dig.*, vol. 2, pp.445-447, June 1999.

[2] Bosui Liu, Yu Albert Wang, George Kang, Altan M. Ferendeci, Missoon Mah, "Vertically interconnected 3D module for conformal phased array antennas", *IEEE Phased Array Symp.*, Dana Point, California, May 2000;

[3] N. I. Dib, R. N. Simons, and L. P. B. Katehi, "New uniplanar transitions for circuit and antenna applications," *IEEE Trans. Microwave Theory and Tech.*, vol. MTT-43, no. 12, pp. 2868-2873, Dec. 1995.

[4] Y. Qian, and T. Itoh, "A broadband uniplanar microstrip-to-CPS transition", *Asia-Pacific Microwave Conf. Dig.*, pp. 609-612, Dec. 1997.

[5] R. Chadha, and K. C. Gupta, "Compensation of discontinuities in planar transmission lines," *IEEE Trans. Microwave Theory and Tech.*, vol. MTT-30, no. 12, pp. 2151-2156, Dec. 1982.

[6] M. Houdart, and C. Aury, "Various excitation of coplanar waveguide," *IEEE MTT-S Int. Microwave Symp. Dig.*, vol. 1,pp. 116-118, 1979.

[7] J.C. Tippet, and R. A. Speciale, "A Rigorous technique for Measuring the scattering matrix of a multiport device with a 2-port network analyzer", *IEEE Trans. Microwave Theory and Tech.*, vol. MTT-30, no .5, pp.661-666, May 1982.

Progress Report
Jan. 1, March-31, 2001
Submitted by University of Cincinnati

During the months of January-March 2000, in addition to the continuing work on antenna, a wide-bandwidth balun, and post characterization, a set of new MEMS switches were processed and tested. In the mean time, a 2x2 array design was initiated..

MEMS Switches.

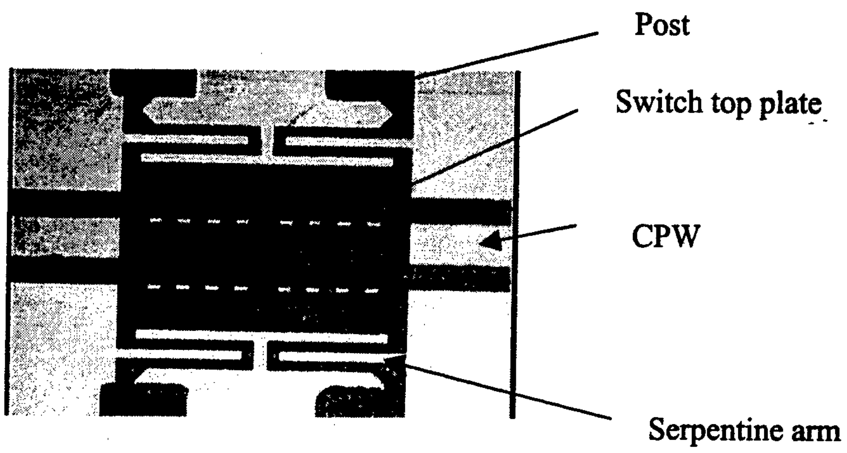


Figure 1. Top view of MEMS switch

A MEMS switch was designed, processed and characterized. Figure 1 show the photograph of the completed switch.

The process steps are:
a) Silicon was used as the substrate
b) Ground plane was deposited (Cr/Cu/Cr) (1.50 μm)
c) Polyimide was spun over the metal (10 μm)
d) Circuit plane was deposited (Cr/Cu) (1.50 μm) and the CPW was processed.
e) SiN2 (1000 A) was deposited and patterned to cover an area larger than the switch top plate.
f) Photo resist was deposited, patterned for the post openings,
g) The same photoresist was used as a sacrificial layer and metal cover (Cu) including the posts was deposited and patterned.
h) Sacrificial layer was removed and switch was tested.

Overall switch length was 400 μm
The S parameters were measured using HP-8510C vector network analyzer using the probing station. The pull down voltages for the switches varied between 10-12 V. The novelty of these switches were that only Copper metal was used for the switch and were processed over polyimide dielectric. The switch on-off times and power handling capability is being measured.

Figure 2 shows the S11 and S21 when the switch was in the up position. Figure 3 show the same parameters at the switch down position.

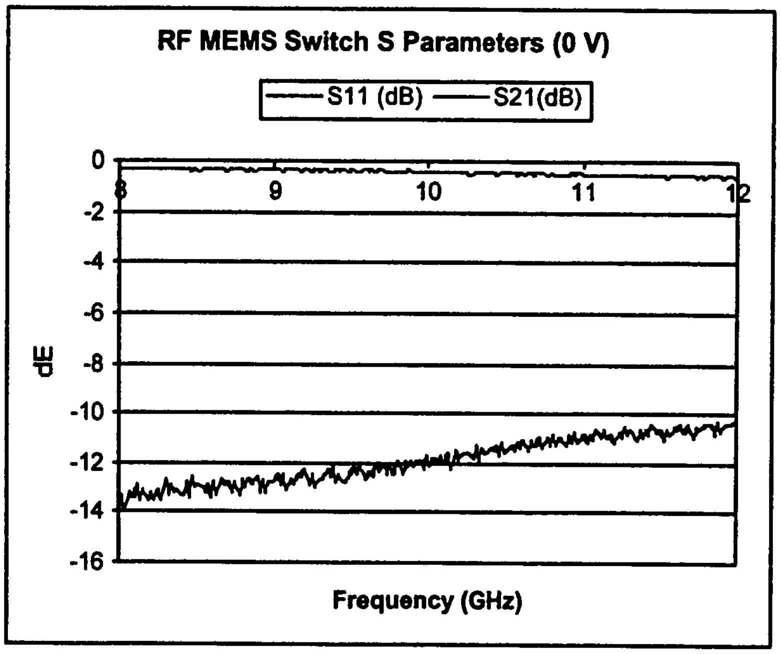


Figure 2. S parameters of the switch in the up position.

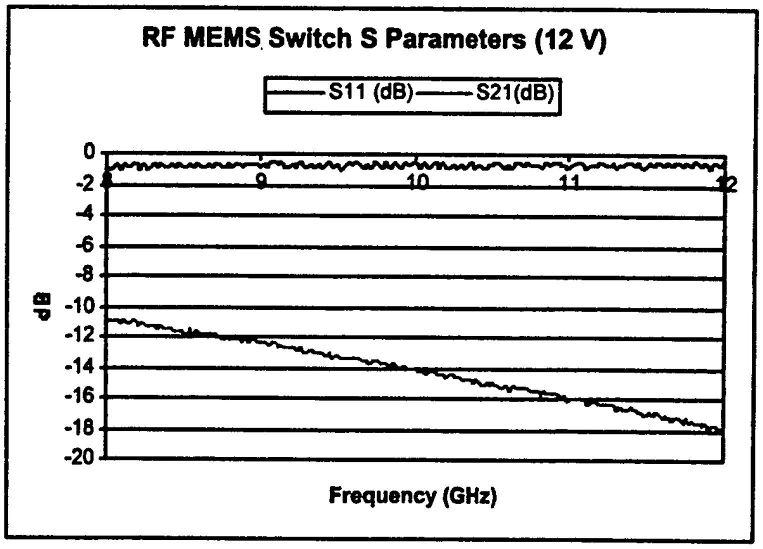


Figure 3. S parameters of the switch in the down position

2x2 Array

A 2x2 array is being developed as a demonstration of the 3D-IC concept. The system will have various components developed by different partners of this projects. Figure 4 below show the general concept of the 2x2 array. It will consist of two wafers. One containing the 4 spiral slot antennas and the second containing the PA, LNA and MEMS switches. The two will be wire bonded to form a single unit. Each antenna will contain a spiral slot antenna and a balun for the feed. For this demonstration, because of the signal distribution requirements, no phase shifting for the elements will be incorporated. This will be left for further development.

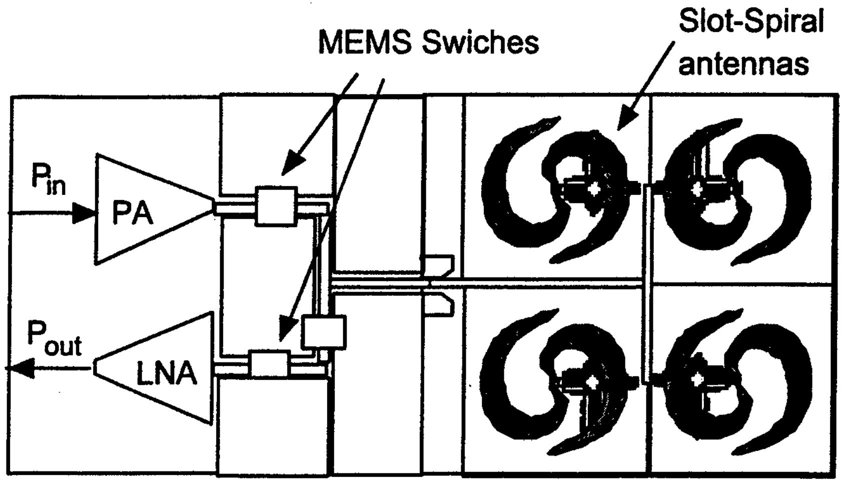


Figure 4. 2x2 planar array

Papers presented

[1] 3D-IC Hybrid Power Amplifier for Vertically Interconnected Multilayer Phased Array Antenna Module.* Altan M. Ferendeci, Peng Xu, and Misoon Mah, GOMAC 2001, San Antonio Texas, (March 5-8, 2001)

*This work is supported by the US Air Force, Air Force Research Laboratory, Contract Number: F33615-96-2-1945

TASK 19

Distributed Simulation of Mixed-Technology for Joint Battlespace Infosphere

Principal Investigator:	Profs. Harold W. Carter and Philip A. Wilsey
Institution:	University of Cincinnati, Cincinnati, Ohio
Period of Performance:	June 20, 1999 to June 10, 2002
Task:	Investigate, develop, and demonstrate techniques to support large distributed simulation of mixed-technology systems (systems containing both discrete and continuous elements) in support of grand challenge Air Force modeling and analysis needs. The simulation kernel should be organized as a modular infrastructure suitable for supporting new application layers (i.e., modeling environments) to maximize generality and reusability. Develop and implement a collaborative infrastructure to support team-based research and education.

Distributed Simulation of Mixed-Technology for Joint Battlespace Infosphere

Final report for Task 19 of Cooperative Agreement #F33615-96-2-1945

Principal Investigators: Philip A. Wilsey and Harold W. Carter
Dept of ECECS, PO Box 210030
University of Cincinnati
Cincinnati, OH 45221-0030

August 9, 2002

1 Project Overview

This project is for research activities by Professors Philip A. Wilsey and Harold W. Carter at the University of Cincinnati, Ohio, and several graduate research students to perform research in two areas: (1) application of distributed discrete-event simulation to major critical needs of the Joint Battlefield Infosphere, and (2) development of a collaborative infrastructure to support team-based research and education. A review of the project objectives and tasks follows.

Objectives

1. Establish state-of-the-art research modeling, simulation & design activities with the Information Directorate and DAGSI schools in the area of distributed simulation of mixed-technology.
2. Integrate new projects into the CERC and II Consortium activities for future DARPA funding and workshop symposiums.
3. Enhance current Air Force modeling and analysis techniques for C4ISR systems.

Tasks

1. Investigate, develop, and demonstrate techniques to support large distributed simulation of mixed-technology systems (systems containing both discrete and continuous elements) in support of grand challenge Air Force modeling and analysis needs. The simulation kernel should be organized as a modular infrastructure suitable for supporting new application layers (i.e., modeling environments) to maximize generality and reusability.
2. Develop and implement a collaborative infrastructure to support team-based research and education.

2 Project Accomplishments

With respect to the project objectives, we have achieved the following accomplishments:

1. *Experiments with simulating models containing more than 10 million objects.* In these experiments we have augmented the WARPED simulation kernel with additional features for ultra-large scale simulation. These extensions include support for out-of-core simulation, state and object reuse capabilities. In the extreme we were able to run a 10 million node simulation on an 8 node cluster where each node was a dual processor Pentium II (300MhZ) containing only 128MB of RAM. The simulation ran poorly, but it did operate correctly. We feel that these experiments show that we can easily run simulations containing more than 10 million objects on a moderate sized Beowulf cluster. A paper [1] documenting the results from this study is attached to this report.

2. *Experiments with component substitution to reduce simulation complexity for higher simulation throughput.* In these experiments, we performed experiments to develop techniques to simulate large, high resolution models by enabling *dynamic* (*i.e.*, during simulation) tradeoffs between model resolution and simulation performance. Such tradeoffs can be achieved by dynamically changing the "level of abstraction" of selected parts of the model. Using dynamic abstraction techniques, the resolution of the model can be dynamically altered to suit the needs of the simulation study – scenarios of interest (or parts of the model) can be simulated in high resolution while the remainder of the simulation (or the model) can be simulated in low resolution. In other words, the simulation proceeds using a lower resolution model until a particular scenario of interest (as defined by the user) is reached; at that time the model is transformed to a higher resolution equivalent for more details; and then model is reverted back to a lower resolution until the next scenario of interest is reached. Such techniques fall under the broad umbrella of Multi-Resolution Modeling (MRM), Cross Resolution Modeling (CRM), and Variable Resolution Modeling (VRM).

   While this work is ongoing, the details of some of our achievements can be found in our attached publications [2,3,4]. Our work with dynamic component substitution has been in a web-based simulation environment where the modeler builds models on a remote client and the simulation is performed on a large server cluster. The environment is build using OO design techniques with well-developed APIs to ease model development and analysis. Currently we are working on mechanizing the abstraction/refinement aspects of the simulation to trigger on issues other than strict performance. For example, in large scale network analysis, modelers are searching for what are called "rare events" to study how the system behaves when they occur. We are now studying this problem to discover if it is practical to introduce a programmable condition that can detect when such events (or other happenings) occur and trigger a refinement of the simulation to a more detailed model.

3. *Integrate mixed-technology modeling and simulation capabilities into the SAVANT/TyVIS/*WARPED *design tools.* We have managed to expand the SAVANT analyzer with an object to help it support the multiple dialects of VHDL. We have introduced VHDL-AMS parsing capabilities into the SAVANT analyzer. We have also created a VHDL (and dialects) test suite that is compliant to the POSIX test standard using the `dejagnu` test harness. The test suite is called VESTs and we have populated it with tests from various sources. Specifically, we have received permissions to include the following in VESTs: (i) the Billowitch test suite (VHDL

'93), (ii) the VHDL '93 models from Peter Ashenden's book on VHDL, and (iii) the VHDL-AMS tests developed at UC under the SEAMS and SIERRA projects. We are currently working with Morgan Kaufmann to obtain permission to add the VHDL-AMS models from his forthcoming book on VHDL-AMS to VESTs.

3 Attachments

The following is a list of the publications documenting the technical results from the research performed in this activity. These papers are also attached to this report.

1. D. M. Rao and P. A. Wilsey, "An Ultra-large Scale Simulation Framework," *Journal of Parallel and Distributed Computing.* (in press).
2. D. M. Rao and P. A. Wilsey, "Performance Prediction of Dynamic Component Substitutions," *Proceedings of the 2002 Winter Simulation Conference*, December 2002. (forthcoming).
3. D. M. Rao and P. A. Wilsey, "Improving Efficiency of Network Simulations through Dynamic Component Substitution," *Proceedings of the Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS 2001), August 2001.
4. D. M. Rao and P. A. Wilsey, "Dynamic Component Substitution in Web-Based Simulation," *Proceedings of the 2000 Winter Simulation Conference*, December 2000.

An Ultra-large Scale Simulation Framework[1]

Dhananjai M. Rao and Philip A. Wilsey
Experimental Computing Laboratory Department of Electrical & Computer Engineering and Computer Science University of Cincinnati, Cincinnati, OH 45221-0030.
E-mail: dmadhava@ececs.uc.edu, philip.wilsey@ieee.org

Many modern systems involve complex interactions between a large number of diverse entities that constitute these systems. Unfortunately, these large, complex systems frequently defy analysis by conventional analytical methods and their study is generally performed using simulation models. Further aggravating the situation, detailed simulations of large systems will frequently require days, weeks, or even months of computer time and lead to scaled down studies. These scaled down studies may be achieved by the creation of smaller, representative, models and/or by analysis with short duration simulation exercises. Unfortunately, scaled down simulation studies will frequently fail to exhibit behaviors of the full-scale system under study. Consequently better simulation infrastructure is needed to support the analysis of ultra-large (models containing over one million components) scale models.

Simulation support for ultra-large scale simulation models must be achieved using low-cost commodity computer systems. The expense of custom or high-end parallel systems prevent their widespread use. Consequently, we have developed an Ultra-large Scale Simulation Framework (USSF). This paper presents the issues involved in the design and development of USSF. Parallel simulation techniques are used to enable optimal time versus resource tradeoffs in USSF. The techniques employed in the framework to reduce and regulate the memory requirements of the simulations are described. The API needed for model development is illustrated. The results obtained from the experiments conducted using various system models with two parallel simulation kernels (comparing a conventional approach with USSF) are also presented.

Key Words: Large Scale Modeling, Parallel and Distributed Simulation, Time Warp Simulation, Unsynchronized Simulation

[1]Support for this work was provided in part by the Defense Advanced Research Projects Agency under contract DABT63-96-C-0055.

1. INTRODUCTION

Modern systems such as microprocessors and communication networks have steadily grown in size and sophistication to meet the ever increasing needs and demands. For example, today's microprocessors are built using a few million transistors [14] and the Internet, a global data network, now connects more than 16 million nodes [15]. These systems involve complex interactions between a few thousand to several million entities. The study and analysis of these systems is necessary in order to effectively design, manufacture, and maintain them [15, 25]. Unfortunately, analytical methods of analysis are insufficient to study these systems and experimental techniques such as computer based simulations are usually employed instead [21, 25]. Furthermore, parallel simulation techniques are employed to enable simulation of large systems in acceptable time frames [15, 21, 25]. Simulation enables explorations of complicated scenarios that would be either difficult or impossible to analyze [15]. Due to its effectiveness, simulation has gained considerable importance and is widely used today.

Validity of the models plays central role in analyzing systems using simulation [23]. The models should reflect the size and complexity of the system in order to ensure that crucial scalability issues do not dominate during validation of simulation results. Many techniques, algorithms, and protocols that work acceptably for small models consisting of tens or hundreds of entities may become impractical when the size of the system grows [15]. Events that are rare or that do not even occur in small toy models may be common in the actual system under study. Detailed simulation of the complete system is necessary to study large scale characteristics, long term phenomena, and to analyze the system as a whole. Paxson *et al*, provide an excellent context from the networking domain to highlight this issue. They write, "Indeed, the HTTP protocol used by the World Wide Web is a perfect example of a success disaster. Had its designers envisioned it in use by virtually the entire Internet – and had they explored the corresponding consequences with experiments, analysis or simulation – they would have significantly altered its design, which in turn would have led to a more smoothly operating Internet today." [15]. Since today's systems involve a large number of entities ranging in the order of a few millions, modeling and simulating such ultra-large systems is necessary.

Simulation of large systems is complicated due to their sheer size. The memory and computational resources needed to simulate such large systems in acceptable time frames are often beyond the limits of a single stand alone workstation [18]. Developing large and complex models while paying special care to optimally utilize system resources (in particular, memory) is a tedious task demanding considerable expertise from the modeler. Parallel simulation techniques need to be efficiently exploited to meet the computational requirements. However, investing in large and expensive hardware components for a "one time" analysis of the system models is seldom economically viable. Hence, simulating large systems using modest hardware resources is an attractive and often the only alternative.

This paper presents the design and evaluation of an Ultra-large Scale Simulation Framework (USSF) that was developed to enable and ease effective simulation of large systems. In particular, USSF was motivated by the need to support analyzing systems involving millions of entities using only modest computational resources.

The framework utilizes parallel simulation techniques to harness the resources of conventional workstations to provide optimal time versus resource tradeoffs. Various software techniques have been employed to reduce and regulate the memory requirements of the simulations. USSF provides a flexible and robust object-oriented API for model development. The API also insulates the model developer from the intricacies of enabling large simulations.

The remainder of this paper is organized as follows. A brief description about the parallel simulation kernels that are used as the underlying synchronization kernels of USSF are presented in Section 2. In Section 3 brief descriptions about some of the earlier research activities related to large scale simulations are presented. Section 4 outlines the software techniques used to alleviate the memory bottlenecks faced while enabling large scale simulations. A detailed description of the USSF along with the API is presented in Section 5. The results obtained from the experiments using the framework with different models and parallel simulation kernels are presented in Section 6. Section 7 provides some concluding remarks with pointers to future work.

2. BACKGROUND

The parallel simulation capability of USSF is enabled by developing the framework around a given parallel simulation engine. That is, the framework runs as an application on an underlying parallel kernel and utilizes its services. Object oriented (OO) techniques have been employed to isolate the various modules of USSF from the underlying simulation kernel. This design was adopted in order to obtain a desired level of "separation of concerns" so that the design of the framework can focus on enabling large scale simulations. The architecture of the framework can be viewed as extending the capabilities of the underlying parallel simulation engine. The design also enables USSF to be easily deployed on different simulation kernels. In this study, USSF was deployed on two different parallel simulation kernels; namely WARPED: an optimistic parallel discrete event simulation (PDES) kernel based on Time Warp [17]; and NoTIME: an unsynchronized PDES kernel [19, 24]. The following subsections provide a brief description about these two simulation kernels.

2.1. WARPED

WARPED is an optimistic PDES kernel that uses the the Time Warp [7] paradigm for distributed synchronization. A Time Warp synchronized simulation is organized as a set of asynchronous logical processes (LPs) that represent the different physical processes being modeled. The LPs exchanging event information by exchanging ***virtual time*** stamped event messages. Virtual Time [9] is used to model the passage of time and defines a total order on the events in the system. Each LP processes its events by incrementing a local virtual time (LVT), changing its state, and generating new events. Although each LP processes local events in their correct time-stamp order, events are not globally ordered. Causality violations are detected when an event with time-stamps lower than the current LVT (a *straggler*) is received. On receiving a straggler event a rollback mechanism is invoked to recover from the causality error. The rollback process recovers the LP's state prior to the causal violation, canceling the erroneous output events generated, and re-processing the events in their correct causal order. Each LP maintains a queue of state transitions along with lists of input and output events corresponding to each state to enable the recovery process. A periodic garbage collection technique based on Global Virtual Time (GVT) is used to prune the queues by discarding history items that are no longer needed. The distributed simulation is deemed to have terminated when all the events in the system have been processed in their correct causal order.

The WARPED kernel presents an interface to build logical processes (LPs). The kernel provides an Application Program Interface (API) to build different LPs with unique definitions of state. The basic functionality for sending and receiving events between LPs using a message passing system is supported by the kernel. In WARPED, LPs are placed into groups called "clusters". LPs on the same cluster communicate with each other without the intervention of the message passing system, which is faster than communication through the message passing system. Although LPs are grouped together into clusters they are not coerced into synchronizing with each other. Control is exchanged between the application and the simulation kernel through cooperative use of function calls. Further details on the API and working of WARPED is available in the literature [10, 11, 17].

2.2. NOTIME

NoTime is an unsynchronized PDES kernel [19, 24]. Unsynchronized simulations have been successfully employed for simulation of stochastic models such as queueing models and communication networks. NoTime simulations provide considerable improvements in performance, when compared to WARPED simulations, with negligible loss in accuracy of the simulation results. A brief description of NoTime is presented in the following paragraph and further discussions on unsynchronized simulations are available in the literature [19, 24].

The NoTime PDES kernel provides necessary support to develop applications modeled as discrete event simulations. NoTime mirrors the API utilized by WARPED. The design enables models developed for WARPED to be run using NoTime without changes to the application. Similar to WARPED, the LPs are grouped into clusters. Processor level parallelism occurs at the cluster level and each cluster is responsible for communication management and scheduling the LPs contained by the cluster. NoTime utilizes the Message Passing Interface (MPI) libraries for communicating between the parallel clusters. Communication between LPs on the same cluster occur without intervention of the communication layer. Since the parallelism occurs at the cluster level, simulation objects that execute relatively independent of each other can be placed on different clusters to maximize parallelism. Conversely, simulation objects that frequently communicate with each other should be placed on the same cluster to exploit the fast intra-cluster communication. Each cluster uses a single input queue that contains the events for all the LPs associated with it in order to optimize scheduling and intra-cluster communication. NoTime uses a First-In-First-Out (FIFO) scheduling scheme. The parallel simulation terminates when all the events in the system have been processed. The kernel uses a circulating token scheme for termination detection. Additional details on NoTime are available elsewhere [19, 24].

3. RELATED RESEARCH

This section presents some of the other techniques that have been employed to enable large scale simulations. Simulation of large scale models has received considerable attention in the past. Various combination of software and hardware techniques have been used to improve the capacity and efficiency of simulators. Huag *et al* present a novel technique to selectively abstract details of the network models and to enhance performance of large simulations [6]. Their technique involves modification of the network models in order to achieve abstraction [6]. Premore and Nicol present issues involved in development of parallel models in order to improve performance [16]. In their work, they convert source codes developed for ***ns***, a sequential simulator into equivalent descriptions in Telecommunications Description language (TeD) to enable parallel simulation [16]. Coupled with meta languages (such as TeD) [16], parallel network libraries and techniques to transparently parallelize sequential simulations have been employed [22]. Riley *et al* present a federated approach to enable parallel simulation in order to improve the capacity for simulating large models. In their technique, existing sequential simulators are extended to enable parallel simulation using conservative synchronization techniques [22].

Relaxation and even elimination of synchronization, a large overhead in parallel simulations, has been explored [19, 26]. The relaxation techniques attempt to improve performance at the cost of loss in accuracy of the simulation results [19, 24]. Martini, Rümeskasten and Tölle [12] propose a novel synchronization protocol called "tolerant synchronization" in which a conservative synchronization protocol is allowed to optimistically process events that are within a tolerance level. As the protocol optimistically processes events, causality violations may occur but these violations are ignored and no recovery process is initiated to rectify the errors. Fall exploits a combination of simulation and emulation in order to study models with large real world networks [4]. This method involves real time processing overheads and necessitates detailed model development. Carothers *et al* present a novel technique called "Reverse Computation" that eliminates the need for state saving in Time Warp simulations and reduces the memory requirements of the simulations [3]. However, not all computations can be reversed and this technique necessitates development of additional simulation code to undo computations.

On the other hand, USSF employs a different approach to enable simulation of large systems using resource constrained platforms. The framework exploits the presence of replicated modeling constructs and the reuse of component (or LP) descriptions to reduce the actual size of the simulations. Replicated structures are identified through static analysis of the model. The framework utilizes a single copy of each unique component (or LP) to mirror its different replicated instances. This approach reduces the overall size of the simulation which in turn reduces the resource consumption of the simulation. USSF also employs a number techniques to reduce and regulate (*i.e.*, improve efficient utilization of main memory) the overall memory requirements of the simulation in order to improve performance. Furthermore, the techniques employed in the framework are independent of the underlying synchronization mechanism and can be applied to any discrete event simulator. Use of the framework does involves only minor changes to the application modules - development of applications is straightforward and it does not require any change in the modeling methodology. In other words, existing applications can be easily ported to exploit the features of the USSF. A more detailed description of USSF is available in the following sections.

4. APPROACH

The initial exploratory simulation studies of large systems were conducted using WARPED [18, 20]. The studies indicated that the memory requirements of such large simulations posed the primary bottleneck in enabling large scale simulations. The memory requirements of a parallel simulation can be classified into two main categories; namely (i) the *static* memory requirements, that do not change over the lifetime of the simulation; and (ii) the *dynamic* memory requirements, that continuously change during simulation. The static memory requirements of the simulation arise due to the LP descriptions (application code), the kernel code, and the various data structures that need to be maintained during the lifetime of the simulation. Many of the kernel data structures need to be duplicated at each of the parallel clusters to enable parallel simulation [17]. Hence, irrespective of the number of workstations employed for parallel simulations, the static memory requirements of the kernel, for a given number of LPs, remains almost constant.

The discrete events and the states of the various LPs contribute to the dynamic memory requirements of the simulation. The dynamic memory requirements are governed by the state sizes of the application modules and their event generation characteristics. The static and dynamic memory requirements of the simulation must be reduced in order to enable large scale simulations using modest hardware resources. Reducing the memory consumed by the parallel simulation kernel, by modifying its data structures, is complex due to the intricate mechanics of optimistic parallel simulations [17, 21]. Modification of the data structures used by the kernel would affect its performance and increase simulation time [21, 20]. Hence, techniques to reduce the static memory requirements and regulate the dynamic memory usage of the application were pursued.

The initial hurdle in enabling large scale simulations was the static size of the application models and their data structures. The application development languages (described in Section 6) provided hierarchical constructs that ease specification of large systems. The hierarchical models were statically elaborated (*i.e.*, at compile time) or "flattened" to a single hierarchical level prior to generating the simulatable code (as illustrated in Section 6). The static elaboration (or compile time elaboration) technique did not scale well in terms of the size of the generated code (the volume of generated code was too large), compilation time (the time taken to compile the generated code as unacceptably long), and static size of the resulting executable. Therefore, a runtime elaboration technique was proposed, in order to reduce the static size of the application modules and in turn its static memory requirements. A *runtime elaboration library* (REL) was developed as a part of USSF to ease runtime elaboration. The runtime elaboration technique and the REL are presented in Section 5.1. A comparative analysis between static elaboration and runtime elaboration is also presented in Section 5.1.

During our exploratory studies of large scale simulations, it was observed that many of the LPs share the same descriptions. The LP descriptions (the code to model the functionality of the LPs) were the same but the data and states were different. The model descriptions could be reused by decoupling them from their data

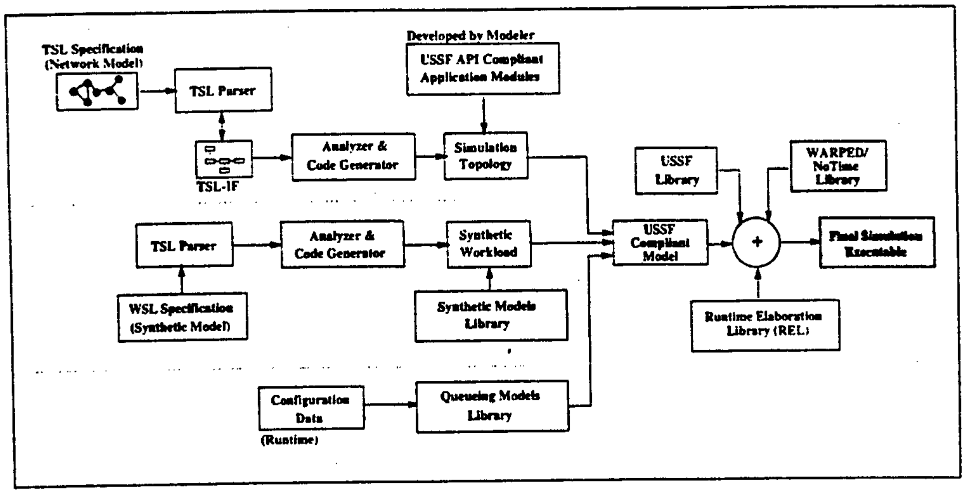

TSL Specification (Network Model)
TSL Parser
Developed by Modeler
USSF API Compliant Application Modules
TSL-IF
Analyzer & Code Generator
Simulation Topology
USSF Library
WARPED/ NoTime Library
TSL Parser
Analyzer & Code Generator
Synthetic Workload
USSF Compliant Model
+
Final Simulation Executable
WSL Specification (Synthetic Model)
Synthetic Models Library
Runtime Elaboration Library (REL)
Configuration Data
(Runtime)
Queueing Models Library

FIG. 1. System Overview

and state. In other words, the same description could be associated with different data and states in order to emulate the various instances of a particular LP. Using this approach a few thousand instances of a given LP could be aggregated into one. This approach reduces the total number of LPs and in turn would reduce the size of the internal data structures of the simulation kernel. Decoupling the data and states also provided a convenient design for swapping data and states in and out of the main memory based on demand, freeing up the memory to contain the discrete events which strongly influence the performance of the simulations [21, 20, 17]. These techniques regulate the dynamic memory consumption of the simulations. The USSF utilizes these techniques to enable effective simulation of large system using limited computational resources. The API of USSF insulates the model developer form the intricacies of enabling ultra-large simulations. A detailed description on the design and implementation of USSF is presented in the following section.

5. THE ULTRA-LARGE SCALE SIMULATION FRAMEWORK

The Ultra-large Scale Simulation Framework (USSF) was developed to ease simulation of large systems. An overview of the framework along with the applications used in this study is shown in Figure 1. As illustrated in the figure the *Networks Simulation Framework* (NSF), the *Performance and Scalability Analysis Framework* (PSAF), and the queueing models generate USSF API compliant code from the corresponding high level model specifications. As mentioned earlier, the compile time elaboration technique used in NSF and PSAF did not scale well when employed for large models involving millions of LPs. Hence, the elaboration technique was modified to employ the runtime elaboration technique (presented in Section 5.1). The necessary *static* analysis to identify and collate the various LPs that share a common object description (as required by USSF API), was also coupled along with code-generation. LPs that share the same description are identified using each object's definition. The object definitions are a part of the high level model specifications and are extracted from the frontend modeling language supported by the application frameworks [21, 1]. The collated information is embedded along with

```
step 1: Initialize elaborator
    1. Initialize new symbol table and IF
    2. Search input IF and locate node corresponding to user specified top level hierarchy
    3. Call elaboration (step 2) with new hierarchy
step 2: Elaboration subroutine (parameter hierarchy)
    1. Process the list of modules specified in the hierarchy. If the module is an object instantiation perform step 3. If the module is a sub-hierarchy perform step 4.
step 3: Elaborate object instantiation
    1. Create new instance of the object instantiation with name-mangled labels.
    2. Create new object definition for the new instance with mangled labels and add to new symbol table, if necessary. Collate information about objects sharing the same object definition.
    3. Add new object instantiation to the new topology and update necessary data structures.
step 4: Elaborate sub-hierarchy
    1. Instantiate a temporary symbol table and IF
    2. Recursively call elaboration with the sub-hierarchy
    3. Implode new sub-hierarchy into current hierarchy
```

FIG. 2. Phases in elaborating a hierarchical design

the generated runtime elaboration code and is utilized by the REL modules. The current implementation of USSF in concordance with WARPED and NOTIME is in C++. Accordingly, the application frameworks also generate code in C++. As shown in Figure 1, the generated code is compiled with the appropriate libraries to obtain the final simulation executable. A brief description about the applications is available in Section 6 and a detailed description about the various components of USSF are presented in the following subsections.

5.1. The Runtime Elaboration Library

Hierarchical language constructs provide convenient techniques to specific large systems by reusing the specification for smaller subsystems [21, 20]. The frontend modeling languages of the application frameworks provide hierarchical constructs to ease specification of large models. However, the hierarchical constructs have to be elaborated or "flattened" prior to simulation. Elaboration is the process in which each hierarchical level is broken down to its constituting components. The basic steps involved in elaborating a hierarchical specification are shown in Figure 2. As illustrated in the figure, the elaborator starts with an user-specified hierarchy and recursively traverses the various components in the model and creates new instances of the sub-hierarchies and the objects. Elaboration of sub-hierarchies is done before they are imploded into the enclosing hierarchy. Imploding hierarchies involves inclusion of all necessary object definitions, object instantiations, and corresponding data structures.

Elaboration may be done *statically* or at *runtime*. Static elaboration occurs prior to code-generation while runtime elaboration occurs prior to simulation, when the generated code is executed. Since static elaboration occurs prior to code-generation,

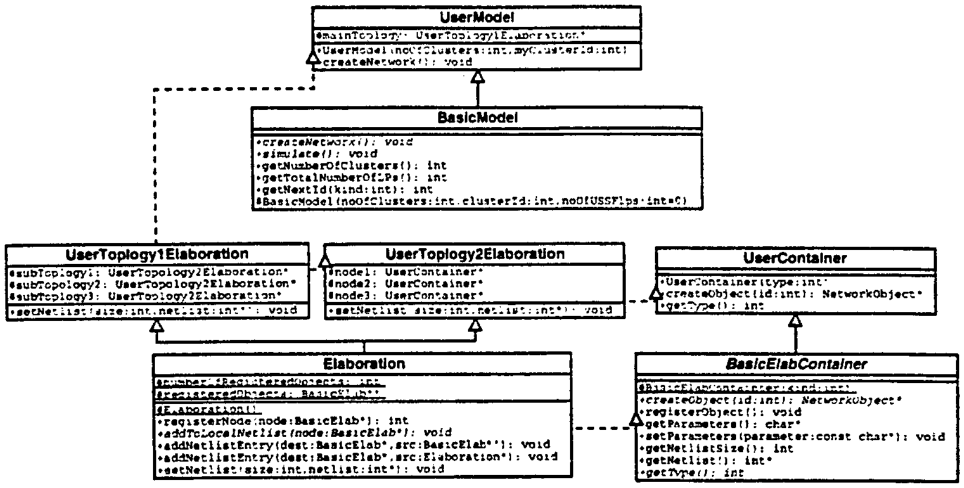

UserModel
#mainTopology: UserTopology1Elaboration*
+UserModel(noOfClusters:int,myClusterId:int)
+createNetwork(): void
BasicModel
+createNetwork(): void
+simulate(): void
+getNumberOfClusters(): int
+getTotalNumberOfLPs(): int
+getNextId(kind:int): int
#BasicModel(noOfClusters:int,clusterId:int,noOfUSSFlps:int=0)
UserToplogy1Elaboration
#subToplogy1: UserTopology2Elaboration*
#subTopology2: UserTopology2Elaboration*
#subToplogy3: UserTopology2Elaboration*
+setNetlist(size:int,netlist:int*): void
UserToplogy2Elaboration
#node1: UserContainer*
#node2: UserContainer*
#node3: UserContainer*
+setNetlist(size:int,netlist:int*): void
UserContainer
+UserContainer(type:int)
+createObject(id:int): NetworkObject*
+getType(): int
Elaboration
#numberOfRegisteredObjects: int
#registeredObjects: BasicElab**
#Elaboration()
+registerNode(node:BasicElab*): int
+addToLocalNetlist(node:BasicElab*): void
+addNetlistEntry(dest:BasicElab*,src:BasicElab**): void
+addNetlistEntry(dest:BasicElab*,src:Elaboration*): void
+setNetlist(size:int,netlist:int*): void
BasicElabContainer
#BasicElabContainer(kind:int)
+createObject(id:int): NetworkObject*
+registerObject(): void
+getParameters(): char*
+setParameters(parameter:const char*): void
+getNetlistSize(): int
+getNetlist(): int*
+getType(): int

FIG. 3. UML diagram for Runtime Elaboration API

the volume of code generated for large models is considerably higher when compared to the code generated when using runtime elaboration. For example, for a network model consisting of a million nodes the number of lines of C++ code generated is a few million. The few million lines of code occupy large volumes of disk space and the compilation times are unacceptably long. As illustrated by the experiments presented in Section 6.4, the static elaboration technique does not scale well. In contrast, the code generated for runtime elaboration merely captures the hierarchical structure of the network model and passes the information to the REL. The volume of the generated C++ code is considerably less and hence compilation times are reduced. The static size of the executable for runtime elaboration is also smaller. However, runtime elaboration involves additional overheads prior to simulation. Runtime elaboration provides a tradeoff between the size of the generated code, the compilation time of the generated code, and overall simulation time.

The *runtime elaboration library* (REL) was developed to ease elaboration of large models. The modules of the REL interact with the various modules of USSF kernel to during elaboration and construct the simulation at runtime. Figure 3 illustrates the important classes that constitute the API of the REL. The `Elaboration` class, the `BasicElabContainer` class, and the `BasicModel` class form the core infrastructure of the REL. The `BasicElabContainer` class is the base class for all the elaboration classes that are used to instantiate the actual LPs that constitute the simulation. For each unique simulation module, the code-generator is responsible for generating a corresponding container class with the `BasicElabContainer` class as its parent. As shown in Figure 3, the generated container class overloads the necessary pure virtual method in the `BasicElabContainer` class. The `Elaboration` class is the base class for each of the generated elaboration classes. For each level in a hierarchical design, an unique class is generated by the code-generator. The generated code contains the necessary calls to the various methods in the `Elaboration` class. On instantiating an elaboration class for a given hierarchy, pointers to the underlying sub-hierarchies and objects, code-generated as member objects, are suitably instantiated. Runtime elaboration proceeds in a depth-first manner. The `BasicModel`

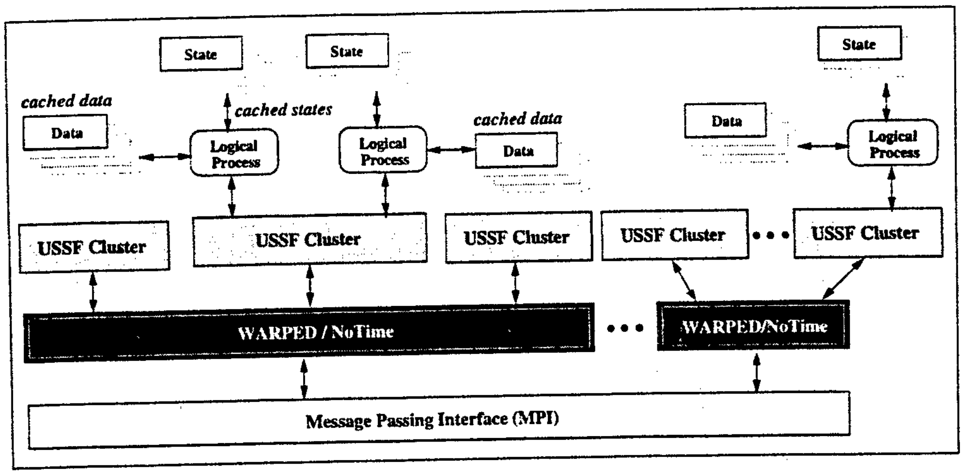


FIG. 4. Layout of an USSF Simulation

class acts as the base class for generating the top level elaboration class. The generated code triggers runtime elaboration by instantiating the elaboration class corresponding to the top most hierarchy in the design. The elaboration data structures constructed in memory are deleted as the recursive decent unwinds in order to ensure minimal memory usage. The REL also includes support for elaborating models directly for WARPED and NOTIME without necessitating any changes to the generated code.

5.2. USSF Kernel

The core functionality of regulating the memory requirements of the application modules are handled by the USSF kernel modules. The kernel modules present an interface similar to WARPED and NOTIME to the model developer. The USSF API is presented in the following subsection. The core of the USSF kernel is the USSF cluster. The USSF cluster represents the basic building block of USSF simulations. Each USSF cluster is assigned and addressed by an unique *id*. A USSF cluster performs two important functions. It not only acts as a LP to WARPED and NOTIME, it also acts as a cluster to the application programmer. As shown in Figure 4, the USSF cluster is used to group a number of LPs that use the same description together. A single copy of an user process is associated with different data and states to emulate its various instances. The USSF cluster uses file based caches to maintain the data and states of the various processes. The caching helps in regulating the demands on main memory. Separate data and state caches are maintained to satisfy concurrent accesses to data and state spaces and to reduce cache misses. On encountering rollbacks, the kernel flushes all the caches to maintain cache consistency and coherence. OO techniques have been used to decouple the various memory management routines from the core of the USSF kernel. This design not only provides a simple mechanism to substitute various memory management algorithms but also insulates the USSF cluster from their intricacies.

The USSF cluster is also responsible for scheduling the various application processes associated with it. The USSF cluster appropriately translates the calls made by the underlying simulation kernel into corresponding application process calls. It

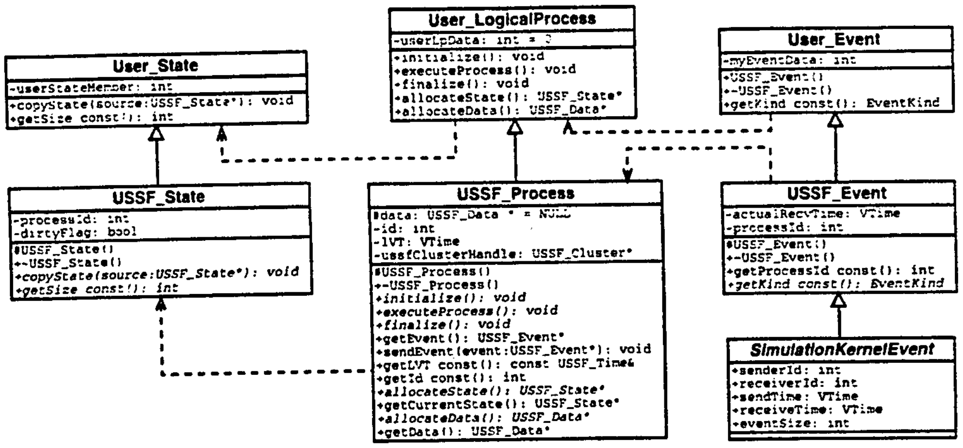


FIG. 5. Overview of USSF API

is also responsible for routing the various events generated by the application to the simulation kernel. The WARPED and NOTIME kernels support exchange of events between the USSF clusters. To enable exchange of events between the various user LPs, the USSF cluster translates the source and destination of the various events to and from USSF cluster *ids*. In order for USSF kernel to perform these activities, a table containing the necessary information is maintained by the kernel modules. The table is indexed using the unique process *ids* that need to be associated with each user LP. To reduce the number of entries in this table, a single entry is maintained for a group of LPs sharing a process description. The static analysis phase of the various application frameworks assigns contiguous *ids* to processes constructed using the same simulation objects. This fact is exploited to efficiently construct and maintain the table. The USSF cluster also maintains a file based state queue in order to recover from rollbacks [7] that could occur in a Time Warp simulation. An incremental state saving mechanism with a fixed (at compilation time) checkpointing interval is used for this purpose [5]. The states space of the USSF cluster contain the corresponding offsets of the checkpoint and state information in the state queue. The offsets are used to restore the states efficiently after a rollback. A simple garbage collection mechanism triggered by the garbage collection routines in WARPED is used to prune the state queues. Access to the various methods in the USSF kernel is provided via a set of application program interfaces, illustrated in Section 5.3. Further details on the design and implementation of the USSF kernel is available in the literature [21, 20].

5.3. USSF Application Program Interface (API)

The API presented by USSF closely mirrors the WARPED API [17]. This enables existing WARPED and NOTIME applications to exploit the features of USSF with few modifications to USSF. The API has been developed in C++ and the object oriented features of the language have been exploited to ensure it is simple and yet robust. The API plays a critical role in insulating the model developer from the intricacies involved with enabling ultra-large parallel simulations. The interface has been carefully designed to provide sufficient flexibility to the application developer

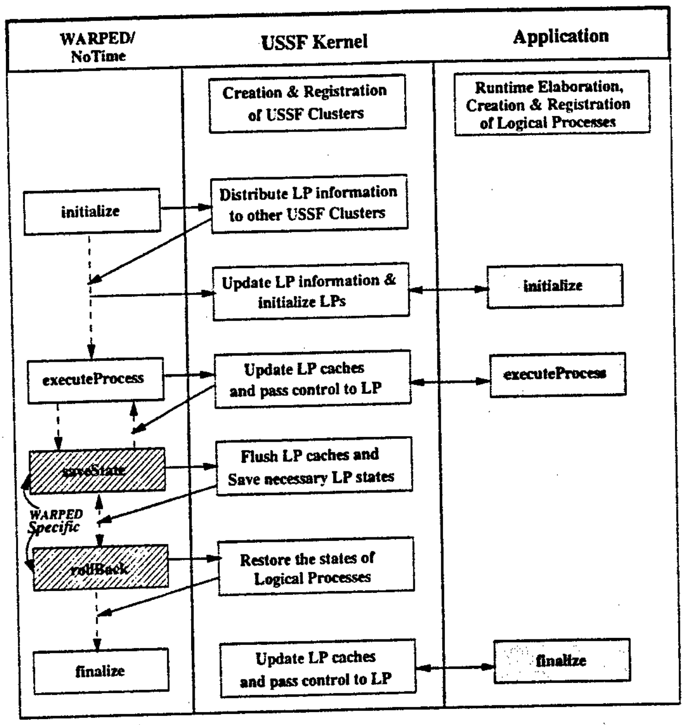

WARPED/ NoTime
USSF Kernel
Application
Creation & Registration of USSF Clusters
Runtime Elaboration, Creation & Registration of Logical Processes
initialize
Distribute LP information to other USSF Clusters
Update LP information & initialize LPs
initialize
executeProcess
Update LP caches and pass control to LP
executeProcess
saveState
Flush LP caches and Save necessary LP states
WARPED Specific
rollBack
Restore the states of Logical Processes
finalize
Update LP caches and pass control to LP
finalize

FIG. 6. Flow of Control in USSF

and enable optimal system performance. Figure 5 presents an overview of the important classes that constitute the API.

The USSF Kernel presents an interface to the application developer for modeling a set of communicating local processes (LPs). The LPs are modeled as entities which send and receive events to and from each other, and act on these events by applying them to their internal state. The `USSF_Process` class form the base class for all the LPs. The basic functionality that the `USSF_Process` class provides for modeling LPs are methods for sending and receiving events between the LPs and the ability to specify different types of LPs with unique definitions of state. The `USSF_State` and `USSF_Event` form the base classes for the states and events in the system, as shown in Figure 5. The user is expected to override some of the kernel methods that are invoked at various times through out the simulation. Each method in this set has a specific function. The `initialize` method gets called on each LP before the simulation begins. This gives each LP a chance to perform any actions required for initialization. The method `finalize` is called after the simulation has ended. The method `executeProcess` of a LP is called by the USSF kernel whenever the LP has at least one event to process. The kernel calls `allocateState` and `allocateData` when it needs the LP to allocate a state or data on its behalf. The various interface classes along with the inheritance hierarchies for specifying application data, states, and events are shown in Figure 5. Although it is the responsibility of the modeler to assign unique *ids* to each LP, the static analysis modules in the USSF perform this functionality. Interfaces for constructing application data, states, and events are also specified. Control is exchanged between the application and the USSF kernel modules through cooperative use of function calls. A detailed flow of control in the system via the API calls is presented in the following subsection.

5.4. Flow of Control in USSF

The flow of control in the USSF is illustrated in order to fully highlight the issues involved in the various aspects of its design. Figure 6 shows the various interactions between USSF, the underlying parallel simulation kernel, and the application during simulation. The first phase of the simulation deals with setting up of the various USSF clusters and the processes contained in them. The runtime elaboration modules perform the task of elaborating the topology and instantiating the necessary LPs. As the processes are created and registered with the USSF cluster, the internal tables are updated. At the end of this phase, the various USSF clusters register themselves with the underlying simulation kernel. The `initialize` method of the various USSF cluster processes are invoked by the underlying simulation kernel. The USSF clusters then exchange time stamped events distributing their internal tables to other clusters. It is important to note that the USSF kernel events have a time stamp that is lower than those of the applications. This is necessary to ensure that the underlying kernels schedule USSF kernel events before the application events are scheduled. Processing the kernel events is crucial in order to ensure that the internal data structures are updated before any of the application's processing begins. Once updating of the internal data structures is complete, the USSF clusters call the `initialize` methods of all the LPs associated with them. When the `executeProcess` method of the USSF cluster is invoked, it updates the data and state caches of the corresponding LP and in turn calls the LP's `executeProcess` method. The events generated by the application are appropriately translated to USSF Cluster *ids* and dispatched using the underlying kernel's interfaces. The USSF cluster also saves the state of the various processes as when the state saving methods are triggered. The saved states are used to restore the states of the various LPs when a rollback occurs during simulation. Garbage collection is done when the routines are triggered by WARPED or NOTIME. Finally, when the `finalize` method of the USSF cluster is called, the USSF cluster calls the `finalize` method of the various LPs associated with it and clears all its data structures and the simulation terminates.

6. EXPERIMENTS

The experiments conducted to evaluate the performance of USSF and the results obtained are presented in this section. All the experiments were conducted on a network of shared memory multi-processor (SMP) workstations. Each workstation consisted of two 300MHz Pentium II processors with 128MB of main memory. The workstations were networked using fast Ethernet. The experimental analysis of the USSF was conducted using three applications; namely the Network Simulation Framework [20], the Performance and Scalability Analysis Framework (PSAF) [1], and queueing models [19]. Figure 1 presents an overview of the interaction between the application frameworks and USSF. The following subsections present brief descriptions about the applications.

6.1. Network Simulation Framework (NSF)

The Network Simulation Framework (NSF) provides a collection of tools to ease modeling and simulation of large scale networks. As shown in Figure 1, the primary input to the framework is the topology to be simulated. The syntax and semantics of the input topology is defined by the Topology Specification Language (TSL), which provides simple and effective techniques to specify hierarchical topologies [18]. A TSL specification consists of a set of inter-connected topologies. A topology consists of a set of interconnected object instantiations (net-lists) along with necessary object definitions. The topology is parsed into an OO Intermediate Format (TSL-IF) using a TSL parser. The parser is generated using the Purdue Compiler Construction Tool Set (PCCTS) [13]. Hierarchical TSL-IFs are elaborated or "flattened" (as illustrated in Section 5.1) prior to code-generation. The elaborated TSL-IF is used to generate necessary C++ code that conforms to WARPED's API for simulation. The NSF in conjunction with WARPED is implemented in C++. The generated topology includes code to instantiate the necessary user defined modules that provide descriptions for the components in the topology. The generated code is compiled along with the WARPED library, and the application program modules to obtain the final simulation executable. Further details on NSF are available in the literature [21, 20].

The network models used in the experiments were constructed by interconnecting a set of subnetworks (representing a local area network) to form a larger network using the hierarchical modeling techniques supported by TSL. The subnetworks were modeled as a set of `nodes` interconnected by a `router`. Each `node` in the network model is driven using a `TrafficGenerator`. The `TrafficGenerator` can be used to generate traffic patterns (such as Constant Bit Rate (CBR)) for modeling different network applications or workloads. Different random number generators based on statistical distributions (such as Poisson distribution and Normal distribution) may also be used to generate network traffic. The `router` component is used to model a simple router (or a switch) in a network. It forwards the packets generated by a `nodes` to the corresponding destination `node` or adjacent `router` as the case may be. Information necessary for routing is established at the time of initialization of simulation. The `routers` build the tables for routing by exchanging information between interconnected `routers`. Interconnections between subnetworks was achieved by suitably interconnecting the `routers`. In the experiments conducted as a part of this study, the `routers` at the higher hierarchical level were interconnected with each other to model different routing domains such as a *stub* domains and *transit* domains [2]. The *transit-stub* network model (used in the experiments) has been shown to be a good model of the Internet [2, 27]. The characteristics of the network models used in the experiments is shown in Table 1.

6.2. Performance and Scalability Analysis Framework (PSAF)

The Performance and Scalability Analysis Framework (PSAF) is a simulation-platform independent tool that can be used to analyze the scalability and performance of any discrete event simulator [1]. The centerpiece of the framework is a platform-independent *Workload Specification Language* (WSL). WSL permits the characterization of simulation model using a set of fundamental parameters that influence the performance of a discrete event simulator [1]. The language provides

TABLE 1
Characteristics of TSL Models used in experiments

Models	Number of Components					Lines of TSL
	Nodes	Traffic Generators	Routers	PacketSinks	Total	
L0N	6	6	2	6	20	53
L1N	30	30	11	30	103	76
L2N	300	300	111	300	1011	99
L3N	3000	3000	1111	3000	10111	122
L4N	30000	30000	11111	30000	101111	145
L5N	300000	300000	111111	300000	1011111	168

constructs that be used to describe synthetic as well as real world benchmarks. PSAF includes a *Synthetic Workload Generator* (SWG) that can be used to generate synthetic benchmarks in WSL. WSL also provides hierarchical constructs that can be used to specify large workloads. The hierarchical constructs provide an efficient technique to scale models to desired proportions using the synthetic constructs of the language. PSAF also provides support to specify platform-specific translation routines that are used to generate a set of simulation models, specific for a given simulation environment, from a WSL description. PCCTS is used to generate the WSL parser. Hierarchical WSL specifications are elaborated (as described in Section 5.1) prior to generation of the simulation models. The generated models can be collectively or individually used as a benchmarking suite to explore the effects of different parameters on the performance of the targeted simulator. The synthetic models generated using PSAF closely reflect the characteristics of several *real world* models [1]. Further details on the design and implementation of PSAF are available in the literature.

The characteristics of the synthetic models used in the experiments is shown in Table 2. These models were developed using the hierarchical WSL constructs. The models consisted of a set of interconnected synthetic LPs with fixed state size and event granularities. A set of event sources and sinks were used to exercise the various LPs constituting the synthetic model. The events were generated using random distributions supported by WSL. The WSL translator was used to generate the synthetic models from the corresponding WSL specification.

6.3. Queueing Models

The queueing models used in this study were built using a library of components developed to explore the usefulness of unsynchronized simulations [19, 24]. The stochastic nature of queueing models make it an interesting case for unsynchronized simulation. Since WARPED and NOTIME present the same API, the queueing models developed using the library can be simulated using either of the parallel kernels without necessitating any changes to the models. The primary components of the queueing library are random sources, queues, servers, and statistics collectors. The components can be used to model any $\mathcal{G}/\mathcal{G}/m$ queueing system [24]. The $\mathcal{G}/\mathcal{G}/m$

TABLE 2

Characteristics of WSL Models used in experiments

Models	Number of Components			
	Sources	Sinks	LPs	Total
WSL1	10	10	80	100
WSL2	100	100	800	1000
WSL3	1000	1000	8000	10000
WSL4	10000	10000	80000	100000

queueing systems represent a generic class of queueing systems that play a central role in a number of physical systems [8]. In such a queueing system the distribution of inter-arrival times and service times are completely arbitrary. The system has m servers and the order of service is also arbitrary. Accordingly, the queueing library permits different random distributions to be associated with the source and servers. The model and scenario of the queueing system to be simulated is specified through configuration files. The queuing models used in the experiments are presented in Section 6.6.

6.4. Comparison between Static and Runtime Elaboration

The statistics obtained from the experiments conducted to evaluate static and runtime elaboration techniques, using the network models shown in Table 1, are presented in Table 3 and Table 4 respectively. The values shown in the tables are average values that were computed using the statistics from ten test runs. Figure 7 shows a comparison between the time taken for code-generation and compilation using static versus runtime elaboration techniques. The memory usage of the TSL parser was monitored by overloading the `new` and `delete` class of C++. As shown in Figure 7, the time for static elaboration, code generation, and compiling the generated code for small network models is lower than the time for runtime elaboration. As the size of the network model increases the time for code generation and compilation increases exponentially with respect to the number of nodes in the

TABLE 3

Statistics on Static Elaboration

Model	Parsing Time (sec)	Elaboration Time (sec)	Peak Memory Usage (KB)	Code Gen. Time (sec)	Lines of C++	Compile Time(sec)
L0N	0.00290	0.0079	10	0.00142	173	3.142
L1N	0.00349	0.0150	29	0.00396	733	4.319
L2N	0.00400	0.3993	267	0.03250	7103	19.832
L3N	0.00475	62.1024	2809	0.32100	70803	957.079
L4N	0.00563	7698.5800	29880	3.72000	707803	N/A

network. The time for compiling the generated code for network model L4N consisting of 100,000 network components is not shown since the C++ compiler ran out of memory while compiling the generated code. Static elaboration technique was not employed for the model LN5 (consisting of more than a million nodes) since the technique failed for a considerably smaller network model (LN4 network model).

In contrast, the corresponding times for runtime elaboration are considerably smaller. The time for generating code for runtime elaboration is small since the network model is not elaborated prior to code generation. The time for compiling the code generated for runtime elaboration is lower since the size of the generated code (in terms of number of lines of C++ is shown in Table 4) is smaller. The reduction in the size of the generated code also reduces the size of the static executable. The drawback of runtime elaboration is that elaboration must be done each time the simulation is run. The time taken for runtime elaboration for the various models is shown in Table 4. Runtime elaboration occurs in each of the parallel clusters. Hence, irrespective of the number of parallel clusters used in the simulation, the time for runtime elaboration remains the same for a given network model.

6.5. Comparison between WARPED and USSF

The network and synthetic models shown in Table 1 and Table 2 were simulated using WARPED and USSF with WARPED as the underlying simulation kernel. A very aggressive GVT computation was used so as to ensure rapid garbage collection to reduce memory consumption of the simulations. Runtime elaboration was utilized for simulating the models. The simulations were run in parallel using a varying number of processors. The LPs were randomly partitioned onto the parallel clusters. The simulation times of the network models shown in Table 1 are shown in Figure 8(a) and Figure 8(b). The parallel simulation times for the synthetic models using WARPED are shown in Figure 9(a) and Figure 9(b). The graphs also shown the time for simulating the models using a sequential simulator. The sequential simulations were conducted using the sequential simulator that is available as a part of WARPED. The sequential simulator also uses WARPED's API and hence the models were run using the sequential kernel without any changes. The sequential simulator could not be used to simulate the large models (such as L4N, L5N,

TABLE 4
Statistics on Runtime Elaboration

Model	Parsing Time (sec)	Elaboration time (sec)	Peak Memory Usage (KB)	Code Gen. Time (sec)	Lines of C++	Compile time(sec)
L0N	0.00290	0.021	1.33	0.0040	340	6.062
L1N	0.00349	0.031	1.99	0.0052	380	6.641
L2N	0.00400	0.123	2.66	0.0073	482	7.675
L3N	0.00475	1.480	3.33	0.0257	584	9.286
L4N	0.00563	15.800	4.66	0.2080	686	10.749
L5N	0.00626	65.010	4.92	2.0140	788	12.244

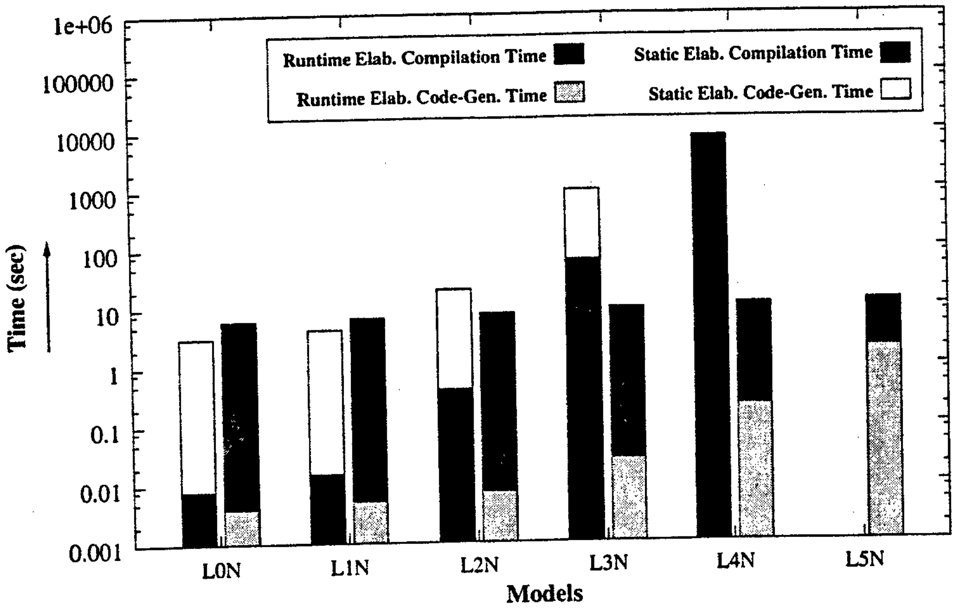
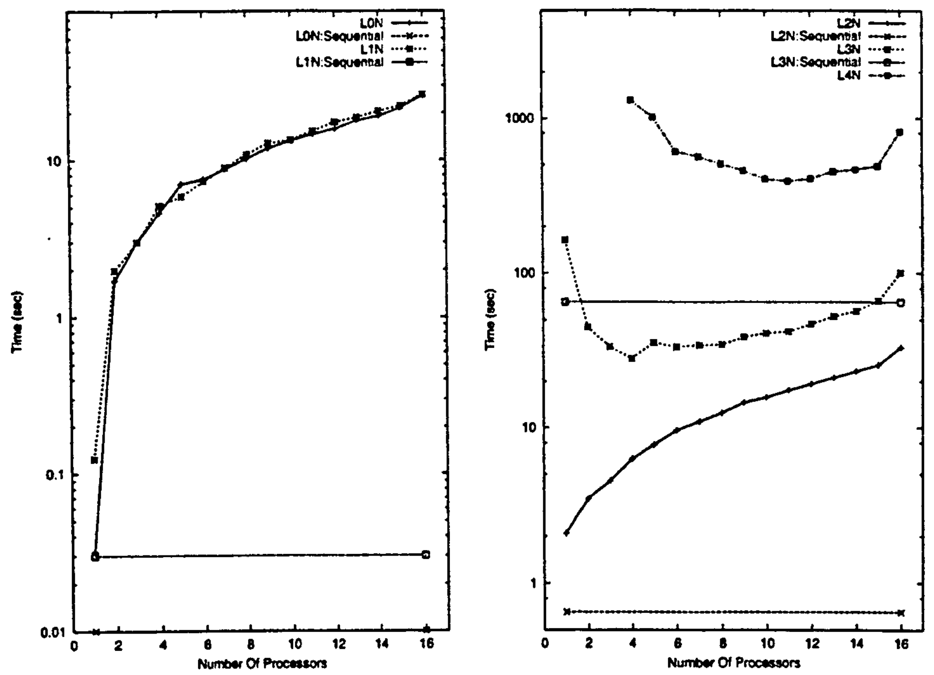


FIG. 7. A comparison between certain Static and Runtime Elaboration Parameters

and WSL4) as the simulations exceeded the memory limits of the systems and did not complete successfully. Figure 10(a) and Figure 10(b) present the corresponding simulation times of the models obtained using USSF. The data plotted in the graphs are the average simulation time values computed from 10 simulation runs.

As illustrated by the graphs shown in Figure 8, Figure 9, and Figure 10; for both WARPED and USSF the simulation time for small models (such as L0N, L1N, WSL1, and WSL2) increases as the number of processors utilized in the simulation are increased. The increase in simulation time is due to the limited amount of parallelism available in the small models. Since parallelism is limited, increasing the number of processors utilized in the simulations merely increases the overheads of parallel simulation and the overall simulation times increase. In the case of medium sized models (such as L2N, L3N, WSL3, and WSL4) the performance improves as the number of processors are increased up to a certain threshold where the gains accrued by parallel simulation out weight the overheads. Beyond the threshold point, the overheads of parallel simulation dominate as the number of processors are increased and the performance deteriorates. However in the case of the large models (such as L4N and WSL4) the simulation time improves as the number of processors are increased. The improvement in simulation time occurs because the models are large (consisting of 100,000 LPs or more) and and sufficient amount of workload and parallelism is available to exploit the parallel processors. The simulation times for the large models using few processors is not shown either because the simulations took unreasonably long time or they did not complete as they exceeded the memory limits of the system. For example the L4N model could not be run with WARPED using fewer than 4 processors and the L5N model (consisting of more than a million LPs) could be run only using USSF on 16 processors.

The graph in Figure 11(a) presents a comparative picture between the simulation execution times of small models using WARPED and USSF. Figure 11(b) presents

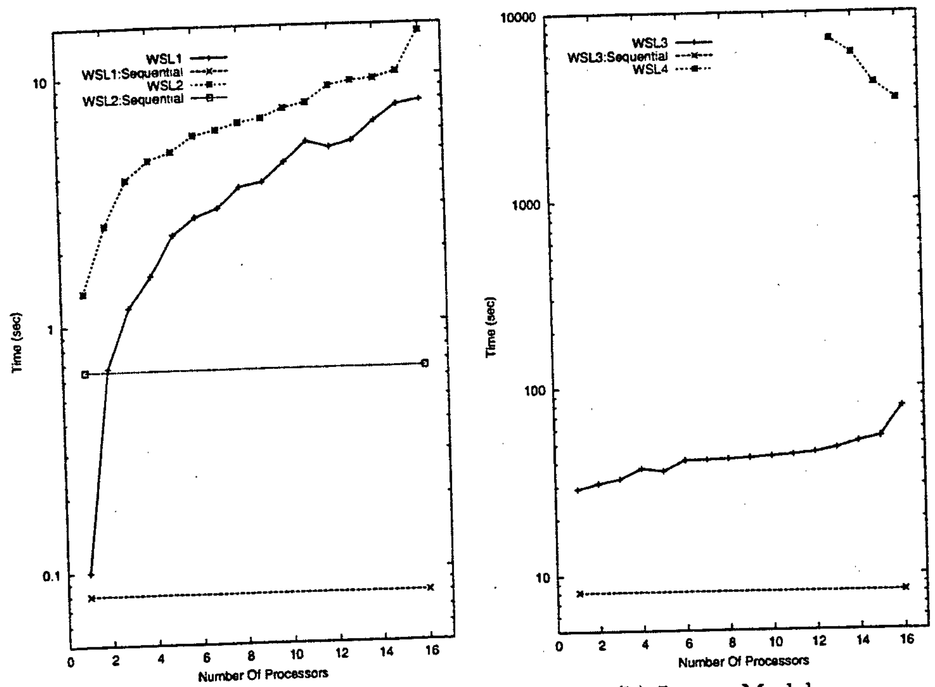


(a) Smaller Models (b) Larger Models

FIG. 8. Simulation time for TSL models with WARPED

the peak dynamic memory utilization of the models. The dynamic memory consumption of the simulations was measured by tracking the memory allocated and deallocated by overloading the C++ `new` and `delete` function calls. The peak memory shown in the graph represents the maximum of the memory consumed by any of the WARPED clusters. As illustrated by the graphs, the memory consumption of WARPED is will within the memory limits of the system and the WARPED performs better than USSF. The simulation times with USSF is higher due to additional overheads of USSF. Some of the USSF simulations performed better than WARPED simulations due to fewer rollbacks. The USSF simulations experienced fewer rollbacks because the overheads of the framework inherently throttle the simulation curtailing the aggressiveness of the optimistic simulations. It was also observed that the performance of USSF simulations deteriorate more rapidly than WARPED simulations as the number of rollbacks increase. The two primary factors that increase the cost of rollbacks were: (i) since a number of LPs are aggregated into a singe USSF cluster, for each rollback the cluster experiences, all the LPs in the cluster need to be rolled back to ensure consistency of the simulations; and (ii) since incremental state saving is used the overheads of restoring the state after a rollback is higher when compared to that of WARPED. The observation indicates that aggregation of LPs into USSF clusters results in a trade off between memory usage and simulation overheads. Although aggressive aggregation decreases memory usage it increases simulation overheads.

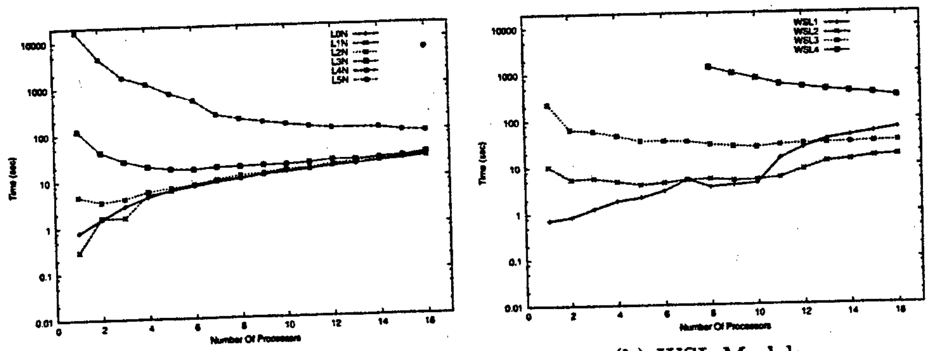
The graphs in Figure 12(a) and Figure 12(b) presents a comparison between the simulation time and peak dynamic memory consumption of the WARPED and USSF



(a) Smaller Models (b) Larger Models

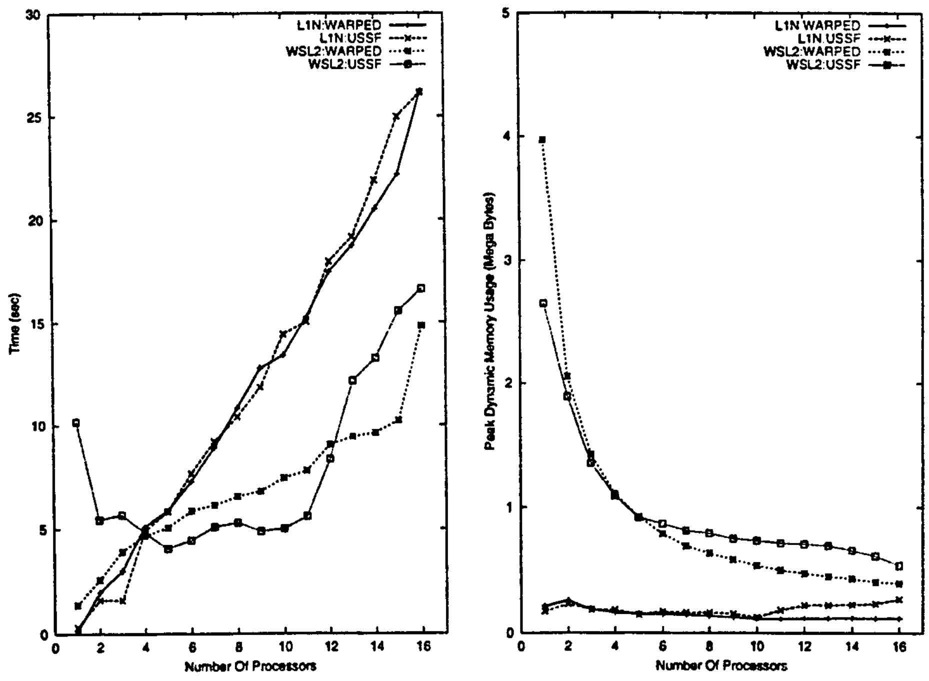
FIG. 9. Simulation time for WSL models with WARPED

simulations. As shown by Figure 12(b) the memory consumption of the simulations steadily decrease as the number processors increases since the memory requirements gets distributed across the processors. As illustrated by the graphs, the simulation times using WARPED and USSF are comparable when the memory consumption of WARPED falls well within the physical memory limits of the workstations used in the simulations. For example, WARPED simulation the L4N model on 4 processors has a peak memory consumption of about 90MB. In contrast, the peak memory requirements of the USSF simulations were considerably lower (about 68 MB) and



(a) TSL Models (b) WSL Models

FIG. 10. Simulation time with USSF

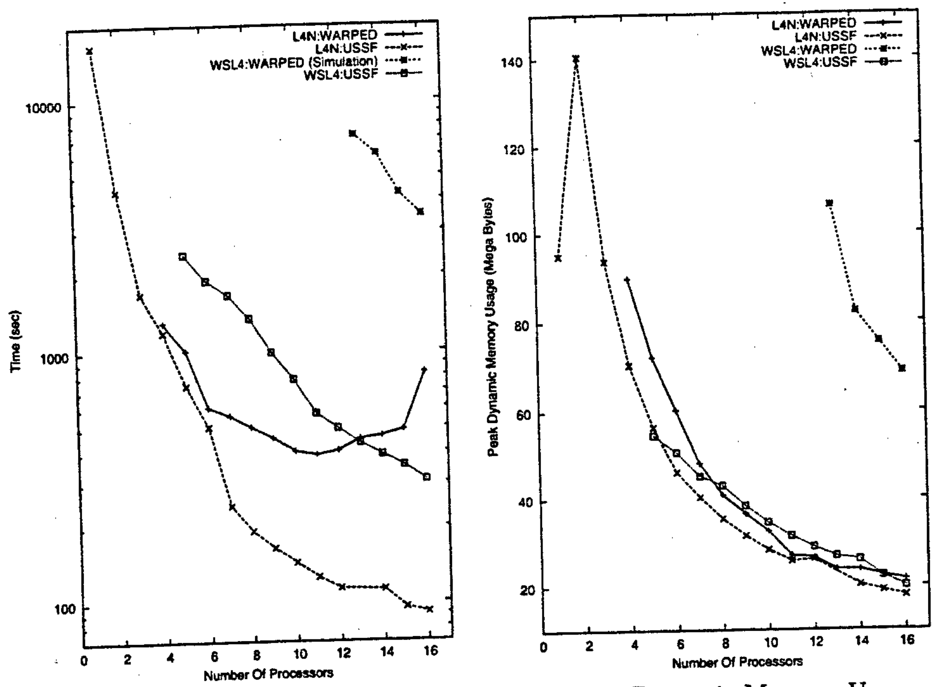


(a) Simulation Time (b) Peak Dynamic Memory Usage

FIG. 11. Simulation Time and Memory Usage of WARPED & USSF for smaller models

hence they perform better. However, as the number of processors are increased the size of the simulations fall well within the physical memory limits of the system and the performance of WARPED simulations steadily increase.

As shown by the graph in Figure 12(a), although the memory requirements of WARPED drops well below the memory limits of the workstations, as the number of processors are increased, the performance of WARPED simulations do not considerably improve. An analysis of the various parameters that influence the performance of the parallel simulations revealed that the primary factor that contributed to increase in simulation times was the cost of the initialization phase of the simulations. The initialization cost was high because of the overheads involved in distributing LP information to the various parallel clusters. For each LP, the WARPED kernel sends an event to all the other clusters in the simulation providing them the necessary information. For example in the case of the L4N model, consisting of 106,444 LPs and running on 16 processors, the 16 clusters exchange a total of 1,596,660 MPI messages. Hence, for large models the number of MPI messages used in the simulations increase as the number of processors are increased and hence the communication overheads dominate the initialization phase. On the other hand, since the USSF kernel collapses a large number of user-defined LPs in a single USSF cluster (or a WARPED process), the number of WARPED LPs are few and the number of MPI events used are few. Therefore, the USSF simulations have considerably smaller startup times compared to the WARPED simulations. Consequently, as shown in Figure 12(a), the overall simulation time for USSF simulations is considerably lower than that of WARPED simulations.



(a) Simulation Time (b) Peak Dynamic Memory Usage

FIG. 12. Simulation Time and Memory Usage of WARPED and USSF for larger models

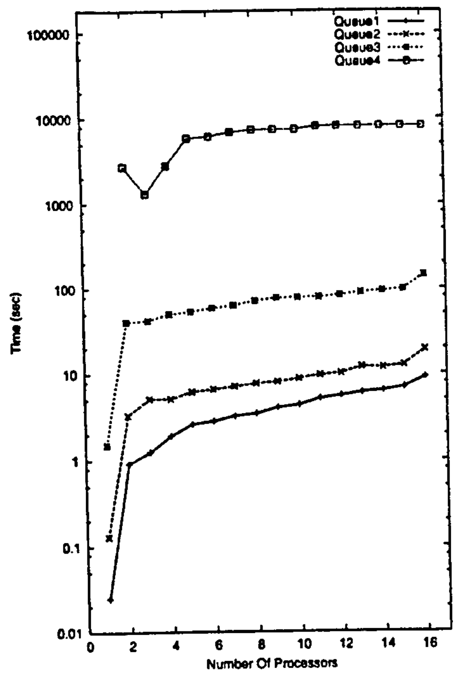
TABLE 5

Characteristics of Queueing Models used in the experiments

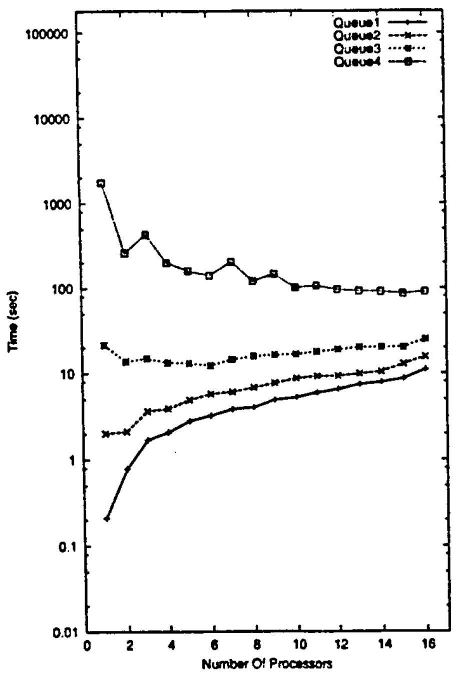
Models	Number of Components					
	Event Generators	Sources	Queues	Servers	Statistic Collectors	Total
Queue1	30	31	4	40	1	106
Queue2	300	312	41	410	1	1063
Queue3	3000	3122	4110	411	1	10644
Queue4	30000	31222	41110	4111	1	106444

6.6. Comparison between NOTIME and USSF

The experiments with USSF using NoTIME as the underlying parallel simulation kernel were performed using different queueing models. The queueing models were built in a hierarchical fashion by randomly cascading generic queueing systems to form larger systems. The queuing models were developed using the queueing model library described in Section 6.3. The runtime elaboration library was used to ease development of the queueing models. Table 5 presents the characteristics of the queuing models used in the experiments. The various queueing models were simulated in parallel using a varying number of processors on a network of workstations. The LPs were randomly partitioned onto the parallel clusters. Figure 13 and Fig-



(a) Using NoTime

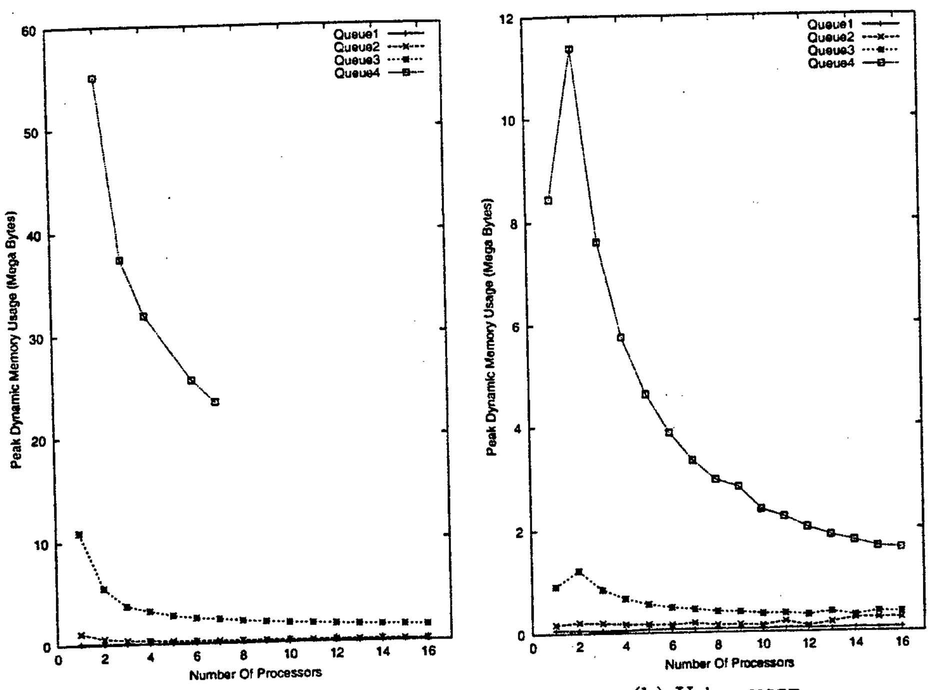


(b) Using USSF

FIG. 13. Simulation times for Queueing models using NoTime & USSF

ure 14 present the simulation times and peak dynamic memory usage of the various queueing model simulations performed using using, respectively, NoTime and USSF (with NoTime as the underlying simulation kernel). The data plotted in the graphs are the average simulation time values computed from 10 simulation runs.

As illustrated by Figure 13(a) and Figure 13(b), the parallel simulation times using USSF and NoTime for small queueing models (such as Queue1 and Queue2) increase as the number of process are increased. However, for the larger queueing model, namely Queue4, the performance of parallel simulations using `ussf` improves as the number of processors are increased. The statistics reflect the nature of the computation to communication ratio of the queueing models. In the case of small models, communication dominates computation, and hence the communication costs increase as the number of processors are increased and the overall simulation time increases. In the case of the large models, computation dominates communication. Hence, as the number of processors are increased the computational overheads are distributed across the parallel processes and the simulation time decreases. The NoTime simulation data for the Queue4 model on one processor is not shown since the simulation exceeded the memory capacity of the workstation and did not complete successfully. The peak dynamic memory requirements of the simulations are shown in Figure 14. As illustrated by the graphs, the memory consumption of the USSF simulations is considerably lower than the memory consumption of the NoTime simulations. The significant difference in memory usage between 1 and 2 processor USSF simulations is indicative of the additional overheads necessary to enable parallel simulation.



(a) Using NoTime (b) Using USSF

FIG. 14. Peak dynamic memory usage using NoTime & USSF

Similar to the initialization overheads of WARPED, the initialization phase of NoTime simulations involved exchanging a large number of messages to distribute information on the configuration of the simulations. Hence, the time for initializing NoTime simulations increases as the number of processors are increased. In the case of NoTime the cost of initialization phase is more pronounced due to the reduced overhead of the simulation kernel. However, since the USSF kernel collapses a number of user-defined LPs into a few NoTime LPs, the number of messages exchanged during initialization is reduced. Therefore, as illustrated by the graphs in Figure 13, the initialization time and overall simulation time for USSF simulations (using NoTime) are considerably smaller than those with NoTime.

7. CONCLUSIONS AND FUTURE WORK

The steady growth in size and complexity of modern systems has required their simulation with modest hardware resources to enable detailed yet cost effective study and analysis. An Ultra-large Scale Simulation Framework (USSF) was developed to ease simulation of ultra-large models with limited hardware resources. The issues involved in the design and implementation of USSF were presented in this paper. The techniques used to reduce the static and dynamic memory requirements of large simulations were presented. An API for the runtime elaboration library was presented. Runtime elaboration was shown to out perform static elaboration for large models. A comparison between the performance of USSF using two parallel simulation kernels, namely WARPED and NoTime, with raw WARPED and NoTime simulations were presented. The experiments conducted indicate that USSF simulations perform better for large models. The experiments also demonstrate the capacity of the framework for simulating ultra large systems using resource constrained platforms.

The USSF provides a tradeoff between memory requirements and simulation overheads by varying the number of LPs aggregated into each USSF cluster. LPs that share a common description are aggregated together. Therefore, the number of LPs that share a common description is a critical factor that determines the overall efficiency of the solution provided by USSF. USSF is an ideal candidate for simulating large applications which contain a number LPs that share a common description. Such models are typical in the domains of network modeling and Very Large Scale Integrated-Circuits (VLSI) design. It must be noted that USSF is a general purpose discrete event simulation framework and does not place restrictions on the nature of the discrete event model being simulated. It is also independent of the underlying synchronization mechanism.

The design and development of the ussf is a part of an ongoing research to improve the efficiency of large scale simulations. Further studies are underway to improve the efficiency of ussf. Research is being conducted to determine an optimal level of aggregation for each model based on the availability of hardware resources. Techniques to dynamically (*i.e.,* during the course of simulation) change the degree of aggregation and the number of USSF clusters used in a simulation are also being investigated. The effectiveness of USSF to enable large scale simulations using conservative simulation techniques needs to be explored. Application of the techniques, used in USSF, for simulation of large scale mixed technology system also provides an excellent avenue for further research.

REFERENCES

1. BALAKRISHNAN, V., FREY, P., ABU-GHAZALEH, N., AND WILSEY, P. A. A framework for performance analysis of parallel discrete event simulators. In *Proceedings of the 1997 Winter Simulation Conference* (Dec. 1997).
2. CALVERT, K., DOAR, M., AND ZEGURA, E. W. Modeling internet toplogy. *IEEE Communications Magazine* (June 1997).
3. CAROTHERS, C. D., PERUMALLA, K. S., AND FUJIMOTO, R. M. Efficient optimistic parallel simulations using reverse computation. In *Proceedings of the thirteenth workshop on Parallel and distributed simulation, PADS'99* (May 1999), pp. 126–135.
4. FALL, K. Network emulation in the Vint/NS simulator. In *In Proceedings of the 4th IEEE Symposium on Computers and Communications* (July 1999).
5. FUJIMOTO, R. Parallel discrete event simulation. *Communications of the ACM 33*, 10 (Oct. 1990), 30–53.
6. HUANG, P., ESTRIN, D., AND HEIDEMANN, J. Enabling large-scale simulations: Selective abstraction approach to the study of multicast protocols. In *In Proceedings of International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Networks* (Oct. 1998).
7. JEFFERSON, D. Virtual time. *ACM Transactions on Programming Languages and Systems 7*, 3 (July 1985), 405–425.
8. KLEINROCK, L. *Queueing Systems.* John Wiley & Sons, New York, NY, 1975.
9. LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. *Communications of ACM 21*, 7 (July 1978), 558–565.
10. MARTIN, D. E., MCBRAYER, T., AND WILSEY, P. A. WARPED: A Time Warp simulation kernel for analysis and application development, 1995. (available on the www at `http://www.ece.uc.edu/~paw/warped/`).

11. Martin, D. E., McBrayer, T. J., and Wilsey, P. A. warped: A Time Warp simulation kernel for analysis and application development. In *29th Hawaii International Conference on System Sciences (HICSS-29)* (Jan. 1996), H. El-Rewini and B. D. Shriver, Eds., vol. Volume I, pp. 383–386.

12. Martini, P., Rümekasten, M., and Tölle, J. Tolerant synchronization for distributed simulations of interconnected computer networks. In *Proc of the 11th Workshop on Parallel and Distributed Simulation (PADS 97)* (June 1997), Society for Computer Simulation, pp. 138–141.

13. Parr, T. J. *Language Translation Using PCCTS and C++*. Automata Publishing Company, January 1997.

14. Patterson, D. A., and Hennessey, J. L. *Computer Architecture: A Quantitative Approach*, 2nd ed. Morgan Kaufmann Publishers, Inc, San Mateo, CA, 1996.

15. Paxson, V., and Floyd, S. Why we don't know how to simulate the internet. In *Proceedings of the 1997 Winter Simulation Conference* (Dec. 1997), pp. 44–50.

16. Premore, B. J., and Nicol, D. M. Parallel simulation of TCP/IP using TeD. In *Proceedings of the 1997 Winter Simulation Conference* (Dec. 1997), pp. 437–443.

17. Radhakrishnan, R., Martin, D. E., Chetlur, M., Rao, D. M., and Wilsey, P. A. An Object-Oriented Time Warp Simulation Kernel. In *Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE'98)*, D. Caromel. R. R. Oldehoeft, and M. Tholburn, Eds., vol. LNCS 1505. Springer-Verlag, Dec. 1998, pp. 13–23.

18. Rao, D. M., Radhakrishnan, R., and Wilsey, P. A. FWNS: A Framework for Web-based Network Simulation. In *1999 International Conference On Web-Based Modelling & Simulation (WebSim'99)* (Jan. 1999), A. G. Bruzzone, A. Uhrmacher, and E. H. Page, Eds., vol. 31, Society for Computer Simulation, pp. 9–14.

19. Rao, D. M., Thondugulam, N. V., Radhakrishnan, R., and Wilsey, P. A. Unsynchronized parallel discrete event simulation. In *Proceedings of the 1998 Winter Simulation Conference* (Dec. 1998), pp. 1563–1570.

20. Rao, D. M., and Wilsey, P. A. An object-oriented framework for parallel simulation of ultra-large communication networks. In *Proceedings of the the Third International Symposium on Computing in Object-Oriented Parallel Environments* (Nov. 1999).

21. Rao, D. M., and Wilsey, P. A. Simulation of ultra-large communication networks. In *Proceedings of the Seventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (Oct. 1999), pp. 112–119.

22. Riley, G. F., Fujimoto, R. M., and Ammar, M. H. A generic framework for parallelization of network simulations. In *Proceedings of the Seventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (Oct. 1999), pp. 128–135.

23. Robinson, S. Simulation model verficiation and validation: Increasing the users' confidence. In *Proceedings of the 1997 Winter Simulation Conference* (Dec. 1997).

24. Thondugulam, N. V., Rao, D. M., Radhakrishnan, R., and Wilsey, P. A. Relaxing causal constraints in PDES. In *13th International Parallel Processing Symposium, (IPPS/SPDP '99)* (Apr. 1999), pp. 696–700.

25. Wilsey, P. A. Modeling, analysis and simulation of computer and telecommunication systems. In *Encyclopedia of Computer Science and Technology*, A. Kent and J. G. Williams, Eds., vol. 41. Marcel Dekker, Inc., 1999.

26. Wilsey, P. A., and Palaniswamy, A. Rollback relaxation: A technique for reducing rollback costs in an optimistically synchronized simulation. In *International Conference on Simulation and Hardware Description Languages* (Jan. 1994), Society for Computer Simulation, pp. 143–148.

27. Zegura, E., Calvert, K., and Bhattacharjee, S. How to model an internetwork. In *Proceedings of IEEE INFOCOM* (Apr. 1996), pp. 594–602.

DYNAMIC COMPONENT SUBSTITUTION IN WEB-BASED SIMULATION*

Dhananjai Madhava Rao and *Philip A. Wilsey*
Experimental Computing Laboratory
Dept. of ECECS, PO Box 210030, Cincinnati, OH 45221–0030.
(email: dmadhava@ececs.uc.edu, phil.wilsey@uc.edu)

ABSTRACT

Recent breakthroughs in communication and software engineering has resulted in significant growth of web-based computing. Web-based techniques have been employed for modeling, simulation, and analysis of systems. The models for simulation are usually developed using component based techniques. In a component based model, a system is represented as a set of interconnected components. A component is a well defined software module that is viewed as a "black box" *i.e.,* only its interface is of concern and not its implementation. However, the behavior of a component, which is necessary for simulation, could be implemented by different modelers including third party manufacturers. Web-based simulation environments enable effective sharing and reuse of components thereby minimizing model development overheads. In component based simulations, one or more components can be substituted during simulation with a functionally equivalent set of components. Such Dynamic Component Substitutions (DCS) provide an effective technique for selectively changing the level of abstraction of a model during simulation. It provides a tradeoff between simulation overheads and model details. It can be used to effectively study large systems and accelerate rare event simulations to desired scenarios of interest. DCS may also be used to achieve fault-tolerance in Web-based simulations. This paper presents the ongoing research to design and implement support for DCS in A Web-based Environment for Systems Engineering (WESE).

1 INTRODUCTION

The marked growth in communication technology and software engineering has resulted in significant growth in the use of the World Wide Web (WWW) [1]. The distributed resources of the WWW have been harnessed together using Web-based computing methodologies [1, 11, 10]. Web-based techniques enable active interaction between interconnected computing systems that can be individually or collectively used to provide a generic set of computational resources. These techniques have transformed the WWW into a giant computational infrastructure [11, 10]. The computational infrastructure of the WWW has been exploited to enable Web-based simulations [7, 11, 10]. Web-based simulation is an effective solution to address a number of issues exacerbating modeling, simulation, and analysis, such as: *(i)* effective sharing and reuse of simulation models developed by different modelers [11, 10]; *(ii)* availability and accessibility of the models [7] without loss of proprietary information [11, 10]; *(iii)* portability and inter-operability of the models [14]; *(iv)* capacity for large scale simulations [6, 10]. Due to its effectiveness, web-based simulations are steadily growing in importance.

In web-based simulation environments, the models for simulations are usually developed using component based modeling techniques [8, 11, 10]. In a component based model, a system is represented as a set of interconnected components [11, 10]. A component is a well defined entity which is viewed as a "black box", *i.e.,* only its interface is of interest and not its implementation. A component could in turn be specified using a set of sub-components. During simulation, each *atomic* component is associated with a specific, well defined software module that implements its behavior and functionality. The software modules could be those implemented by the modeler, available locally, or those obtained via the WWW from other third party model developers [11, 10]. Web-based simulation environments insulate the user from the intricacies involved utilizing third party models and the overheads of distributed simulation. Component based modeling techniques offer a number of advantages [8, 11, 10]. Components are not only useful modeling abstractions but are

*Support for this work was provided in part by the Advanced Research Projects Agency under contracts J–FBI–93–116 and DABT63–96–C–0055.

also convenient units for information exchange over the WWW. Sharing and reuse of components considerably reduces modeling and validation overheads. Component based modeling technique also eases exploration of design alternatives through "plug and play" of components [11, 10]. Hence, it is prevalently used for web-based modeling and simulation [1, 11, 10].

In component based simulation models, one or more components can be substituted by functionally equivalent set of components without altering the basic characteristics of the model. Substituting one or more components with a single component and vice versa is synonymous to varying the level of abstraction of the model. For example, in the case of logic simulations, a structural model of a component could be substituted by its behavioral model and vice versa to change the levels of abstraction. Substitution of components may be done statically, prior to simulation, or dynamically, during the course of simulation. Static component substitution has been employed to address capacity and performance of large scale simulations. Huang *et. al.* present techniques for selectively abstracting different components of network models to improve performance and capacity of network simulations [3]. Rao *et. al.* aggregate components that utilize a common implementation, increasing the capacity of simulators, to enable ultra-large scale simulations [12]. Levelized code compilation techniques, that selectively replace parts of combinatorial logic circuits with equivalent behavioral descriptions, are widely used to improve performance of circuit simulations [15]. The primary drawback of these techniques is that, functionality, observability, and model details cannot be altered during simulation. However, observability and model details are crucial for effectively studying large scale systems.

On the other hand, substituting components during simulation provides a dynamic tradeoff between model details and performance of the simulation. Dynamic Component Substitution (DCS) not only encompasses the utility of its static counterpart but also provides a number of other useful features. DCS enables effective "What-if" analyses and exploration of design alternatives to be carried out during the life time of a simulation. DCS is a novel approach for simulation of "multiple futures" [4]. It is an alternative approach for fast simulations and provides an attractive solution to accelerate rare event simulations [13]. It is an effective technique for debugging and validating large simulations. The technique can also be used to dynamically alter the tradeoffs between resource consumption and model details during simulation. DCS can be used to selectively abstract parts of a model thereby enabling simulation and analysis of large systems in reasonable time frames. The technique can also be used to achieve better fault-tolerance in web-based simulations. However, implementing support for DCS in optimistically synchronized simulations, Time Warp simulations in particular, is a complicated task. This paper presents the issues involved in implementing support for DCS in a Web-based Environment for Systems Engineering (WESE). A brief background on the distributed synchronization mechanism and the simulation kernel used in WESE is presented in Section 2. An overview of WESE is presented in Section 3. The issues involved in implemented DCS in WESE are presented in Section 4. Some of the experiments conducted using the DCS feature of WESE are presented in Section 5. Section 6 presents some concluding remarks along with pointers to future work.

2 BACKGROUND

The distributed simulation capabilities of WESE have been enabled using WARPED, a parallel optimistic simulator. WARPED uses the Time Warp [5] paradigm to achieve distributed synchronization. A Time Warp synchronized simulation is organized as a set of communicating asynchronous logical processes (LPs). The LPs communicate between each other by exchanging discrete *virtual time* stamped events [5]. Virtual Time is used to model the passage of time and define a total order on the events in the system. Each LP processes its events incrementing its local virtual time (LVT), changing its state, and generating new events. The LPs must be synchronized in order to maintain the causality of the simulation; although each LP processes local events in their correct time-stamp order, events are not globally ordered. Causal violations may occur due to the optimistic nature of Time Warp. Causality violations are detected by a LP when it receives an event with time-stamps lower than its LVT (a *straggler* event). On receiving a straggler, a *rollback* mechanism [5] is invoked to recover from the causality error. The rollback process recovers the LP's state prior to the causal violation, canceling the erroneous output events generated by sending out anti-messages, and re-processing the events in their correct causal order [5]. Each LP maintains a list of state transitions along with lists of input and output events corresponding to each state to enable the recovery process. A periodic garbage collection technique based on Global Virtual Time (GVT) [5] is used to prune the queues by discarding history items that are no longer needed. The distributed simulation is deemed to have terminated when all the events in the system have been processed in their correct causal order.

The WARPED kernel presents an interface to build

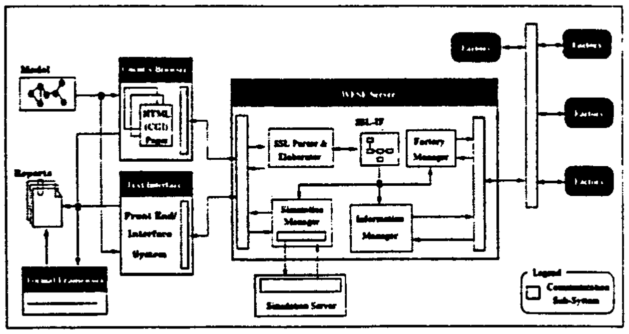


Figure 1: Overview of WESE

logical processes based on Jefferson's original definition [5] of Time Warp [9]. The kernel provides an application program interface (API) to build different LPs with unique definitions of state [9]. The basic functionality for sending and receiving events between LPs using a message passing system is supported by the kernel. In WARPED, LPs are placed into groups called "clusters". LPs on the same cluster communicate with each other without the intervention of the message passing system, which is faster than communication through the message system [9]. Although LPs are grouped together into clusters they are not coerced into synchronizing with each other. Control is exchanged between the application and the simulation kernel through cooperative use of function calls. Further details on the API and working of WARPED is available in the literature [9].

3 WESE

The Web-based Environment for Systems Engineering (WESE) was developed to ease modeling and simulation of systems over the WWW [10]. In WESE the model of a system is represented using a set of interconnected components. A component is treated as a "black box" with a set of inputs and outputs; *i.e.*, only the interface specification of the component is of concern and not its implementation. The actual implementation of a component could be developed by the modeler or by other third party designers. Accordingly, WESE provides a component based modeling language, a framework for developing a web-based repository of components, and the infrastructure for distributed simulation. An overview of WESE is shown in Figure 1. As shown in Figure 1, the environment provides a Hyper Text Markup Language (HTML) interface and a text based frontend that can be used to interact with the WESE Server. The server forms the core of WESE and orchestrates the various parallel and distributed activities of the system. The input to WESE is the model of the system described using the System Specification Language (SSL). The Backus Normal Form for SSL grammar is shown in Figure 2. As shown in Figure 2, the specification of a model or a SSL design file consists of a set of interconnected *modules*. Each module consists of three main sections, namely; *(i)* the *component definition section* that contains the details of the components to be used to specify a module (such as the Universal Resource Locator (URL) of a factory and name of the source object along with initial parameters); *(ii)* the *component instantiation section* that defines the various components constituting the module; and *(iii)* the *netlist section* that defines the interconnectivity between the various instantiated components. SSL permits a *label* to be associated with each module. The *label* may be used as a component definition in subsequent module specifications to nest a module within another. In other words, the *labels*, when used to instantiate a component, result in the complete module associated with the label to be embedded within the instantiating module. This technique can be employed to reuse module descriptions and develop hierarchical specifications. As shown in Figure 1, the input SSL source is parsed into an object oriented (OO) in-memory *intermediate form* (SSL-IF) using the SSL parser. Hierarchical SSL models are elaborated or "flattened" at the end of parsing by the elaborator [11]. Elaboration is a recursive process that flattens a hierarchical model by substituting each module reference (made through the use of *labels*) with an unique instance of the module. As shown in Figure 1, the elaborated model, which is also represented using SSL-IF, forms the primary input to all the other modules of WESE.

The WESE Server also performs the task of collaborating with the distributed factories and coordinating the simulations. As shown in Figure 1, the *simulation manager* performs the activities associated with coordinating with the object factories (via the *factory manager*) to setup a distributed simulation. The *factory manager* performs the tasks of interacting with the distributed factories using a predefined protocol. It not only provides a uniform interface to communicate with different object factories but also insulates the other modules of the server from the intricacies of the underlying protocols. The *information manager* is responsible for interacting with the factories (via the *factory manager*) and constructing the formal specifications used by WESE's formal framework. The current implementation of WESE is geared to generate formal specifications in PVS, a higher order logic specification language. The PVS specification can be used to formally verify different attributes of the system by using a mechanized theorem prover.

To ease design, development, and use of compo-

```
design_file ::=
    include_list ssl_design_module | ssl_design_module
include_list ::= include_clause | include_clause include_list
include_clause ::= include " file_name ";
file_name ::= identifier | identifier . identifier
ssl_design_module ::=
    label ssl_module | ssl_module ssl_design_module |
    ssl_module | label ssl_module ssl_design_module
ssl_module ::=
    { component_definition_section }
    { component_instantiation_section }
    { net_list_section }
label ::=
    identifier (number,number) |
    identifier (number,number label_string )
label_string ::= , identifier | , identifier label_string
component_definition_section ::=
    component_definition |
    component_definition component_definition_section
component_definition ::=
    component_name (number , number):url optional_parameter
optional_parameter ::= parameter ; | ;
component_name ::= identifier (number, number)
parameter ::= " string " | ""
url ::= host_name : port_number . factory
host_name ::= identifier | identifier . host_name
factory ::= identifier | identifier . factory
port_number ::= number
component_instantiation_section ::=
    component_instantiation |
    component_instantiation component_instantiation_section
component_instantiation ::=
    identifier . identifier optional_parameter |    identifier . identifier
number optional_parameter
net_list_section ::= net_list | net_list net_list_section
net_list ::= identifier ( mode , number ) :  instance_list ;
instance_list := instance | instance , instance_list
instance ::= identifier ( mode , number )
mode ::= in | out
identifier ::= start_char any_char
start_char ::= [a - z, A - Z]
any_char ::= [a - z, A - Z, 0 - 9, _]
string ::= string_char | string_char string
string_char ::=  []
number ::= [0 - 9]
```

Figure 2: BNF for SSL grammar

nents WESE provides a framework for constructing web-based *object factories*. An object factory can be viewed as a web-based repository of components with an added capability for simulating them. The object factories play a pivotal role in providing a framework for management of components and the infrastructure for distributed simulation. Figure 3 illustrates the layout of a WESE factory. The initial handle to a factory is provided by the *gateway* module. The module hooks on to a specified IP (Internet Protocol) address via the communication backbone and processes the initial requests from different simulation managers. This IP address that should be specified in the configuration file to locate and communicate with a factory. The task of interacting with a simulation manager to create components and to set up a simulation is handled by the *session manager module*. The session manager also handles some of the specific semantics of the simulation engine. The *configuration manager* tailors the components generated by the factory to meet the user's specifications. The simulation sub-system constitutes the actual simulation engine of the factory. A WESE factory is built from sub-factories and *object stubs*. The object stubs are the atomic components of a factory. Object stubs contain attributes of the physical component (such as cost, size, and speed) along with the formal specifications for the component. The object

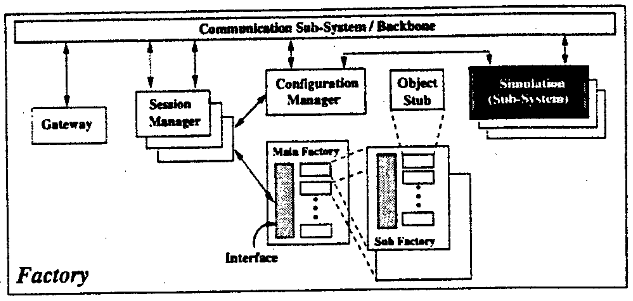


Figure 3: A WESE Factory

factories collaborate with the WESE Server to enable web-based simulations. WESE provides a simple, yet robust *application program interface* (API) for developing simulation models. Further details on the API and WESE are available in the literature [10].

4 IMPLEMENTING DCS IN WESE

DCS may be achieved by replacing a given LP, or a set of LPs, in a simulation with a functionally equivalent LP, or a set of LPs. Some of the scenarios that could arise in DCS are illustrated in Figure 4. The *"1 to 1"* case, shown in Figure 4, in which one LP is replaced by another, is the simplest instance of DCS. As shown in the figure, the *"N to 1"* scenario, where in a set of LPs are replaced with a equivalent LP, arises when a compound component, consisting of a set of sub-components, is replaced with an atomic component. This scenario is equivalent to abstracting a part of the model. The *"N to M"* instance is one where in a set of LPs (N LPs) are replaced with a equivalent set of LPs (M LPs). This scenario arise when a compound component, is replaced with another compound component. However, this instance can be viewed as sequence of atomic component substitutions. An atomic component may be replaced with a compound component, reducing the level of abstraction, causing a single LP

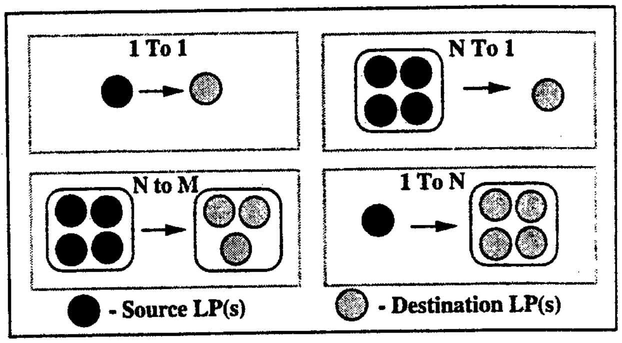


Figure 4: Scenarios in DCS

```
ssl_module ::= auxiliary_component ssl_module_body | ssl_module_body
ssl_module_body ::= { component_definition_section }
    { component_instantiation_section }
    { net_list_section }
auxiliary_component ::= label | : url optional_parameter
```

Figure 5: Modified BNF for SSL Module

to be replaced with a set of LPs. The *"1 to N"* scenario, shown in Figure 4, illustrates this case. To enable modeling of the different scenarios and effectively utilize support for DCS, modifications to the modeling language and simulation infrastructure are required. Consequently, to enable DCS in WESE, modifications to SSL, the SSL parser, SSL-IF, elaborator, and the simulation infrastructure were carried out. The issues involved in the implementation of these modifications along with the tradeoffs in their design are discussed in the following subsections.

4.1 Modifications for modeling DCS

The initial phase of implementing support for DCS in WESE involved extending SSL to include additional constructs for modeling the different scenarios illustrated in Figure 4. Care was taken to ensure that the extensions were minimal so that the language continues to be simple, flexible, and easy to process. The primary extension was to permit an auxiliary *module* or *component definition* to be associated with a module. The BNF of the modified grammar rule for a module is shown in Figure 5. When DCS for a module is requested, the set of components contained by the module are substituted using the auxiliary *module* or *component definition* and vice versa, as the case may be. Modeling the *"N to 1"*, the *"N to M"*, and the *"1 to N"* scenarios (illustrated in Figure 4) using this extension is straightforward. In WESE, DCS can be performed only at a module level. However, a module can contain a single component and it can be replaced it with an auxiliary component. This feature can be exploited for modeling the *"1 to 1"* DCS scenario. The semantics of the *netlist* was also extended to include references to the auxiliary module and component definitions.

SSL-IF was also extended to correspondingly reflect the changes to the grammar. The elaborator was also modified to account for the auxiliary components. The elaborator also flattens auxiliary modules and component definitions. It results in the creation of unique instances of the auxiliary components. The auxiliary components are an integral part of the elaborated SSL-IF and are identified using special flags in the various data structures. The elaborator was extended to identify primary input and output components, *i.e.*, components that are directly connected to the input and output ports of the enclosing module. This information is utilized during simulation to update netlist entries when components are substituted with other components. The elaborator also collates information on the set of components contained by each module. This information is utilized to identify a set of components that need to be replaced when DCS is initiated. The data collated by the elaborator is embedded into the corresponding SSL-IF nodes generated during elaboration. The data is passed on to the simulation modules of WESE that utilize them for enabling efficient DCS. Modifications to the simulation infrastructure of WESE to enable DCS are presented in the following subsection.

4.2 Simulation infrastructure for DCS

The process of dynamically substituting components during simulation (as shown in Figure 4) involves the following steps: triggering DCS in the simulation, creation of new LPs that model the components, updation of states and events of the LPs, and updation of kernel information. In Time Warp synchronized simulations, additional care must be exercised to implement these phases in the presence of rollbacks that could occur in a Time Warp synchronized simulation. A number of modifications were carried out to the simulation modules of WESE to enable DCS. The most significant change was a modification to the structure and API of a LP. The API was modified to utilize object oriented (OO) techniques to completely disassociate a user-defined LP from the simulator core, as shown in Figure 6. In the earlier API, the `UserDefined Object` class would be directly inherited from the `Kernel Object` class. As illustrated in the figure, the `Kernel Object` and `User Object` are linked using pointer references. The `User Object` translates the API function calls to corresponding `Kernel Object` methods while

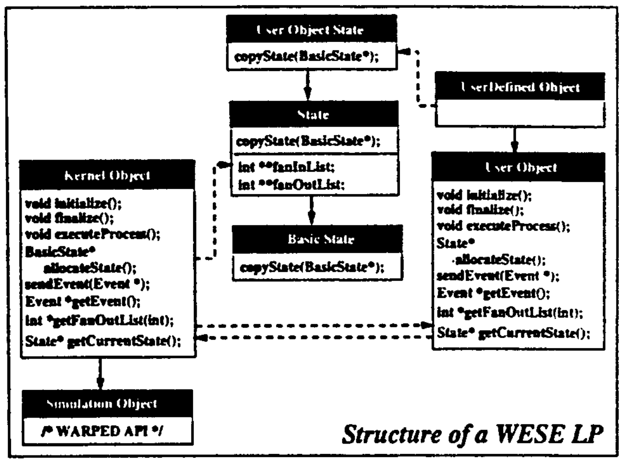


Figure 6: Modified structure of a WESE LP

the Kernel Object translates WARPED API function calls to corresponding User Object methods. The API presented by the User Object class is similar to the earlier API of WESE. Hence, the changes required to the existing components of WESE were minimal.

This design is motivated primarily by two factors. The primary issue being that the WARPED kernel does not support creation and deletion of LPs during simulation. In other words, the WARPED kernel does not permit the structure and composition to change once simulation commences. However, DCS involves changes in structure and composition during simulation. This issue is resolved by using the class hierarchy, shown in Figure 6, wherein the Kernel Objects are static (*i.e.*, they do not change during simulation) while the User Object class hierarchy is dynamic (*i.e.*, it can change during simulation). The Kernel Objects provide the static interface to the WARPED kernel, while different UserDefined Objects can be plugged into the Kernel Object during simulation. This technique enables dynamic substitution of components while adhering to the specifications and semantics of the WARPED kernel. However, this design does not provide an effective technique for creating new components that may be necessary during DCS. Hence, in WESE, the auxiliary components that could potentially be used during simulation are also created. However, these components merely as place holders and do not perform any activity until they are activated through DCS.

The second motivation for the design is that the Kernel Object class provides a convenient spot for implementing support for DCS by utilizing the simulation infrastructure of WARPED. The WARPED kernel insulates the Kernel Objects from rollbacks which considerably reduces the complexity and overheads involved in implementing DCS. Also, with this design, the overheads and process of DCS is transparent to the components. This solution is independent of the underlying synchronization mechanism. Accordingly, in WESE, an event driven approach has been adopted for carrying out the sequence of steps involved in dynamically substituting components. The set of *kernel events* used by WESE was extended to include events for sequencing the different phases of DCS. The primary drawback of this design is that it introduces additional state saving overheads in Time Warp simulations. However, a number of Time Warp optimizations can be employed to minimize state saving overheads [2]. This design also introduces additional overheads during simulation since each API function call involves one extra level of indirection. Also, maintaining the auxiliary components could prove to be a bottleneck for large simulations [12]. However, component aggregation techniques can be employed to minimize the overheads [12].

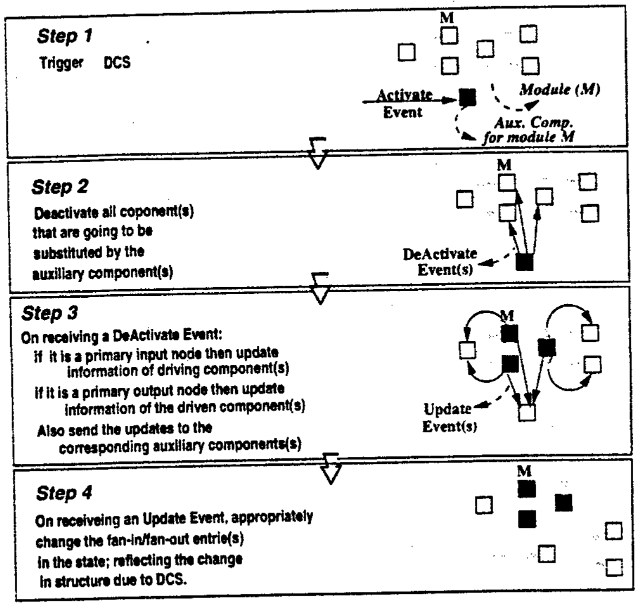

Step 1
Trigger DCS
M
Activate Event
Module (M)
Aux. Comp. for module M
Step 2
Deactivate all coponent(s) that are going to be substituted by the auxiliary component(s)
M
DeActivate Event(s)
Step 3
On receiving a DeActivate Event:
If it is a primary input node then update information of driving component(s)
If it is a primary output node then update information of the driven component(s)
Also send the updates to the corresponding auxiliary components(s)
M
Update Event(s)
Step 4
On receiveing an Update Event, appropriately change the fan-in/fan-out entrie(s) in the state; reflecting the change in structure due to DCS.
M

Figure 7: Sequence of operations during DCS

A typical sequence of steps performed by the Kernel Objects to achieve DCS are shown in Figure 7. The figure also illustrates the corresponding sequence of transformations that occur to the model during the different phases. The kernel events that participate in DCS are also shown. The initial phase involves triggering DCS in the simulation by scheduling an Activate or a DeActivate event, as the case may be, to the corresponding auxiliary component(s). DCS could be triggered externally, by using interactive simulation features, or internally, by the simulation model based on certain application-specific conditions. On receiving a Activate or a DeActivate event, the Kernel Objects initiates the process of DCS. During the second phase of DCS, the activated set of auxiliary components schedule DeActivate events to the set of components that they are going to substitute. The information on the set of components to be replaced is collated during elaboration and is passed onto the corresponding Kernel Objects by the WESE server during initialization. The server also passes the primary input and output component flags collated by the elaborator (as explained in subsection 4.1) along with the netlist data to the respective Kernel Objects. In the next phase, the Kernel Objects that receive the DeActivate Event utilize this information to schedule Update Events to all the related components. The related components are those components with which a given component communicates. This list of related components is obtained from the from the netlist data of the component. The primary input and output components also schedule Update Events to the auxiliary component to

Model Name	Description	Number of Components		
		Regular	Aux.	Replaced by Aux.
M1	4-bit adder	56	4	9
M2	5-bit Mux.	33	6	4
M3	Cascaded half-adders	16	6	2
M4	Chain of not gates	70	5	6
M5	Chain of not gates	330	1	30

Table 1: Models used in experiments

provide the list of related components. This information is required to build the netlist of the new component. On receiving the various `Update Events`, the various `Kernel Objects` update their netlists reflecting the change in structure. As shown in Figure 7, during subsequent simulation cycles, the events generated would be passed on to the new components while the old components get deactivated. To handle the different scenarios that could arise during DCS (as shown in Figure 4, additional sub-tasks, such as instantiating new components in different object factories, are performed in the corresponding DCS phases.

The kernel events used during the different phases of DCS are scheduled using WARPED's simulation infrastructure. The usage is similar to that of any other WARPED application. Hence, to ensure that the events are scheduled in the correct sequence, a *delta* delay is introduced between each event using a two tuple definition for simulation time. The use of a two tuple definition for simulation time is hidden from the user by the API. Since the process of DCS proceeds in *delta* cycles, it appears to occur at a particular instant in simulation time. The *delta* delays also ensure consistent recovery from rollbacks. The data pertaining to the component is stored in the state of the `Kernel Object`. When a rollback occurs, WARPED appropriately restores the state of the `Kernel Object` ensuring coherence of the different phases in DCS. The disadvantages of the event driven design for DCS is that a large number of events could be scheduled during DCS. Hence, if DCS occurs frequently, the performance of the simulation could deteriorate. One of the limitations of the current implementation is that it can be used to substitute only "memory less" components (*i.e.*, components that do not have an explicit notion of state). In other words, the current implementation does not provide support to map the state of the old module to that of the new module. Research is underway to provide a support for mapping the state space of one module to another during DCS. Also, it must be noted that the transient events that were already scheduled for the old set of modules do not get reassigned to the new set of components. They continue to get processed by the substituted set of components. The experiments conducted using DCS in WESE are presented in the following section.

5 EXPERIMENTS

The experiments conducted to evaluate the support for DCS in WESE consisted of two phases. During the first phase an object factory consisting of a collection of logic gates was developed. The factory contained logic gates such as two input `and` gate, two input `or` gate, two input `exclusive-or` gate, and `not` gate. More complex components, such as a `half adder` and a `full adder`, were included in the hardware factory. The factory also contained a bit pattern generation component and a bit display component. The pattern generator can generate all possible bit patterns of a given length and can be used to exercise the inputs of a model developed using components from the factory. The display component can be used to generate a set of bits as outputs from the simulations. The factory also contained a *controller* component that provides a convenient interface to trigger DCS. The second phase of the experiment consisted of developing logic models in SSL using the various components from the hardware factory. The characteristics of some of the models using the experiments is shown in Table 1. The models included auxiliary component specifications for the modules that had equivalent higher level abstractions. The number of components replaced by each auxiliary component in the models is also shown in the table (column `Replaced by Aux.`). For example, model M1 was implemented using structural models of full adders. The structural models of also included an auxiliary specification to use the `full adder` component available in the factory. The `full adder` component substitutes nine components constituting the structural model. The SSL descriptions also used the `controller` components activate the auxiliary modules (trigger DCS) at different time points during simulation.

The simulation experiments were conducted on a network of shared memory multi-processor (SMP) workstations. Each workstation consisted of two Pentium pro Processors (166 Mhz.) with 128 mega bytes (MB) or main memory (RAM). The workstations were interconnected using fast Ethernet. The graph in Figure 8 presents the change in the total number of events processed with respect to the duration of simulation time in which the auxiliary components were active. These statistics were collated from the experiments conducted using a single factory where in no rollbacks occur. The data points shown with zero durations did not involve any DCS and represent the basic number of events executed by each model. As illustrated by the graphs, for short durations during which the auxiliary component

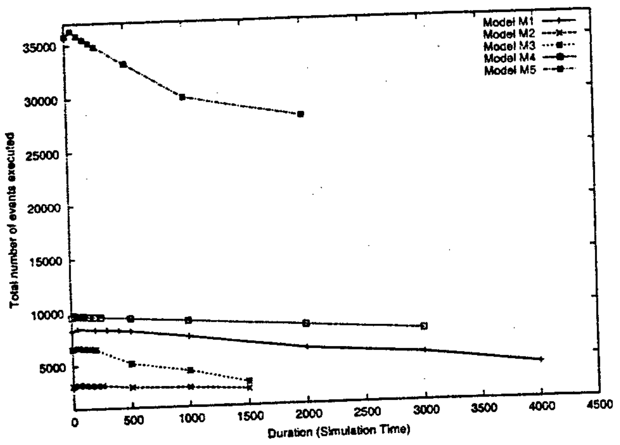


Figure 8: Events versus Duration of DCS

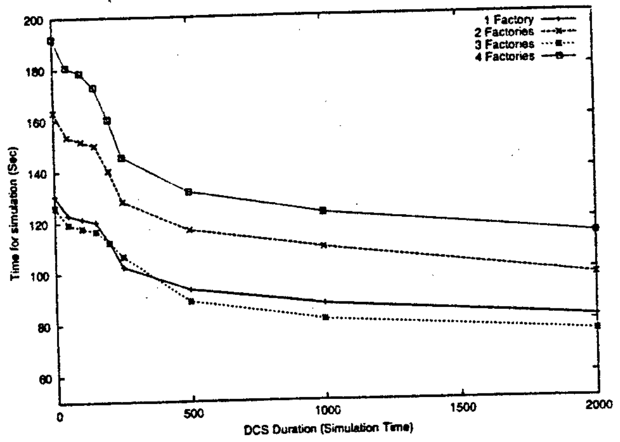


Figure 9: Time for Parallel Simulation

is active, the total number of events processed is higher. The increase in number of events is due to the additional kernel events used to activate and deactivate the components during DCS. However, as the duration increases the number of events processed decreases. The number of events decrease since a set of components are replaced by a single component which results in the elimination of a number intermediate events used and the total number of events in the simulation decreases. As shown in Figure 8, the duration of simulation time for which DCS reduces the number of events varies with respect to the model characteristics. This value plays a crucial role in the effectiveness of DCS to improve performance of the simulations. If the duration is smaller than this threshold value, then as the number of substitutions increases, the total number of events in the simulation increases and the performance of the simulation decreases, and vice versa.

Figure 9 presents the time for simulating model M5 in parallel using a varying number of factories. These experiments were conducted by deploying the object factory on different workstations and modifying the SSL descriptions to choose components from the different factories. The components were chosen from the different factories at random. The timing information shown in the graph is the average of 10 simulation runs. As illustrated by the graph, the performance of the simulations increases as the duration during which the auxiliary components are active increase. As shown in Figure 8, the improvement in performance is due to the decrease in the total number of events that need to be processed. As illustrated by Figure 9, the parallel simulations performed using 3 factories performs better than those performed using a single factory. The performance improves since the simulation overheads get distributed across the three processors. In the 2 factories case the computational overheads dominate the simulation, while in the 4 factories case communication overheads dominate. Hence, in these cases the overheads dominate the gains accrued by employing parallel simulation and the performance of the simulations do not improve. As illustrated by Figure 8 and Figure 9 the performance of parallel simulations can be improved through DCS.

6 CONCLUSIONS

Component based modeling techniques provide a effective means to study systems through "plug and play" of components. In this paper the issues involved in substituting the components dynamically, during simulation were presented. The design and implementation of the support for Dynamic Component Substitution in WESE was illustrated. The experiments in which DCS was used to change the level of abstraction of the model during simulation were described. The results obtained from the experiments indicate that considerable gains in the performance of simulation can be accrued by employing DCS. The technique can be used to accelerate simulations, rare event simulation in particular, to scenarios of interest. DCS can be used to replace a single component with multiple components and simultaneously study the effects of different decisions. This provides a novel technique for simulating multiple futures. DCS can also be used to selectively study parts of a large simulation thereby increasing the performance and the capacity to simulate large scale models over the WWW.

References

[1] FISHWICK, P. A. Web-based simulation: Some personal observations. In *Proc. of the 1996 Winter Simulation Conference* (Dec. 1996), pp. 772–779.

[2] FUJIMOTO, R. Parallel discrete event simulation. *Communications of the ACM 33*, 10 (Oct. 1990), 30–53.

[3] HUANG, P., ESTRIN, D., AND HEIDEMANN, J. Enabling large-scale simulations: Selective abstraction approach to the study of multicast protocols. In *In Proceedings of International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Networks* (Oct. 1998).

[4] HYBINETTE, M., AND FUJIMOTO, R. Optimistic computations in virtual environments. In *Proceedings of the Virtual Worlds and Simulation Conference (VWSIM'99)* (Jan. 1999), pp. 39–44.

[5] JEFFERSON, D. Virtual time. *ACM Transactions on Programming Languages and Systems 7*, 3 (July 1985), 405–425.

[6] PAGE, E. H., GRIFFIN, S. P., AND ROTHER, L. S. Providing conceptual framework support for distributed web-based simulation within the high level architecture. In *Proceedings of SPIE: Enabling Technologies for Simulation Scicence II* (Apr. 1998).

[7] PAGE, E. H., AND NANCE, R. E. Parallel discrete event simulation: A modeling methodological perspective. In *Proceedings of the ACM/IEEE/SCS 8th Workshop on Parallel and Distributed Simulation* (Dec. 1994), pp. 88–93.

[8] PIDD, M., OSES, N., AND CASSEL, R. A. Component-based simulation on the web? In *In Proceedings of the 1999 Winter Simulation Conference (WSC'99)* (Dec. 1999).

[9] RADHAKRISHNAN, R., MARTIN, D. E., CHETLUR, M., RAO, D. M., AND WILSEY, P. A. An Object-Oriented Time Warp Simulation Kernel. In *Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE'98)*, D. Caromel, R. R. Oldehoeft, and M. Tholburn, Eds., vol. LNCS 1505. Springer-Verlag, Dec. 1998, pp. 13–23.

[10] RAO, D. M., CHERNYAKHOVSKY, V., AND WILSEY, P. A. WESE: A Web-based Environment for Systems Engineering. In *2000 International Conference On Web-Based Modelling & Simulation (WebSim'2000)* (Jan. 2000), Society for Computer Simulation.

[11] RAO, D. M., RADHAKRISHNAN, R., AND WILSEY, P. A. FWNS: A Framework for Web-based Network Simulation. In *1999 International Conference On Web-Based Modelling & Simulation (WebSim'99)* (Jan. 1999), A. G. Bruzzone, A. Uhrmacher, and E. H. Page, Eds., vol. 31, Society for Computer Simulation, pp. 9–14.

[12] RAO, D. M., AND WILSEY, P. A. Simulation of ultra-large communication networks. In *Proceedings of the Seventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (Oct. 1999), pp. 112–119.

[13] VILLÉN-ALTAMIRANO, M., AND VILLÉN-ALTAMIRANO, J. Restart: A straightforward method for fast simulation of rare events. In *In Proceedings of the 1994 Winter Simulation Conference (WSC'94)* (Sept. 1994), pp. 282–289.

[14] VINOSKI, S. Corba: Integrating diverse applications within distributed heterogenous environments. *IEEE Communications Magazine, Vol 35, No. 2* (Feb. 1997).

[15] WANG, Z., AND MAURER, P. M. Lecsim: A levelized event driven compiled logic simulation. In *Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC'90)* (June 1990), pp. 491–496.

AUTHOR BIOGRAPHIES

Dhananjai M. Rao is a Ph.D. student in the Department of Electrical and Computer Engineering & Computer Science at the University of Cincinnati. He received his Bachelor's degree in Computer Science and Engineering from the University of Madras in 1996. His research interests include parallel discrete event driven simulation, distributed computing, network simulation, object oriented design patterns, and web-based simulation.

Philip A. Wilsey is an Assistant Professor in the Department of Electrical and Computer Engineering & Computer Science at the University of Cincinnati. He received PhD and MS degrees in Computer Science from the University of Southwestern Louisiana and a BS degree in Mathematics from Illinois State University. His current research interests are parallel and distributed processing, parallel discrete event driven simulation, computer aided design, formal methods and design verification, and computer architecture.

PERFORMANCE PREDICTION OF DYNAMIC COMPONENT SUBSTITUTIONS

Dhananjai M. Rao
Philip A. Wilsey
Experimental Computing Laboratory
Dept. of ECECS, PO Box 210030,
Cincinnati, OH 45221–0030. U.S.A.

ABSTRACT

The Web-based Environment for Systems Engineering (WESE) is a web-based modeling and simulation environment in which the level of abstraction of a model can be configured *statically* (prior to simulation) or *dynamically* (during simulation) by substituting a *module* (set of components) with a equivalent component or vice versa through a process called Dynamic Component Substitution (DCS). DCS can considerably improve the overall efficiency of simulations by enabling dynamic tradeoffs between several modeling and simulation related parameters. However, identifying ideal sequence of DCS is a complicated task. This paper proposes a novel methodology called *DCS performance prediction methodology* (DCSPPM) to identify ideal sequences of DCS. DCSPPM utilizes estimates of the changes induced by each *atomic* DCS along with model characteristics to predict the changes induced by a combination of substitutions. Our studies indicate that the proposed methodology provides good estimates (maximum error $< 8\%$) of the changes induced by DCS.

1 INTRODUCTION

Web-based simulations are steadily growing in importance because they are an effective solution to address several issues exacerbating modeling, simulation, and analysis of modern systems (Rao and Wilsey 2000). To ease web-based modeling and distributed simulation, a Web-based Environment for System Engineering (WESE) has been developed (Rao and Wilsey 2000). WESE provides a hierarchical, component-based modeling language called the System Specification Language (SSL). In SSL, a system is represented as a set of interconnected components. A component is a well defined *atomic* entity which is viewed as a "black box" — *i.e.*, only its interface and functionality is of interest and not its implementation. A set of components that model a sub-system can be grouped into a *module*. Modules are the hierarchical building blocks of SSL. They can be reused (through a well-defined interface) in a hierarchical fashion to develop larger systems. Modules can be viewed as components at a higher level of abstraction. In addition to WESE, component-based modeling techniques are also used in other tools because they offer several advantages (Rao, Chernyakhovsky, and Wilsey 2000).

In WESE, a model may be transformed to a functionally equivalent model by substituting a module (*i.e.*, a set of components) with a *equivalent component* or vice versa. The equivalent component of a module must satisfy the following criteria: (i) it must have an interface that is identical to that of the module, and (ii) its functionality must be similar to that of the module. In this work, we do not deal with issues of establishing equivalence of a module with a component. Instead we leave the decision of equivalence to an oracle which may or may not be the modeler. In other words, when a equivalent component is specified for a module, WESE assumes that it satisfies the necessary criteria. Substituting a module with its equivalent component or vice versa is synonymous to varying the level of abstraction and the resolution of the model (Rao and Wilsey 2000). Figure 1 shows different transformations that can be applied to a typical full adder (digital logic). For example, Figure 1(b) illustrates the modules `exclusive-or gate` and `2-bit mux` (shown in Figure 1(a)) substituted with equivalent components. Transformations to a model can be performed *statically* or *dynamically*. Static transformations occur prior to simulation while dynamic transformations occur during simulation.

In WESE, static and dynamic transformations to a model are effected through a process called Dynamic Component Substitution (DCS) (Rao and Wilsey 2000). DCS can be used to enable optimal, dynamic tradeoffs between several interrelated modeling and simulation parameters such as: modeling costs, resolution of the model, accuracy of results, and simulation performance (Rao, Chernyakhovsky, and Wilsey 2000; Rao, Wilsey, and Carter 2001). It has shown to be an effective technique to improve the overall efficiency of a simulation study (Rao and Wilsey 2000; Rao, Wilsey, and Carter 2001). For example, parts of a model that are inconsequential to a given study can be abstracted in or-

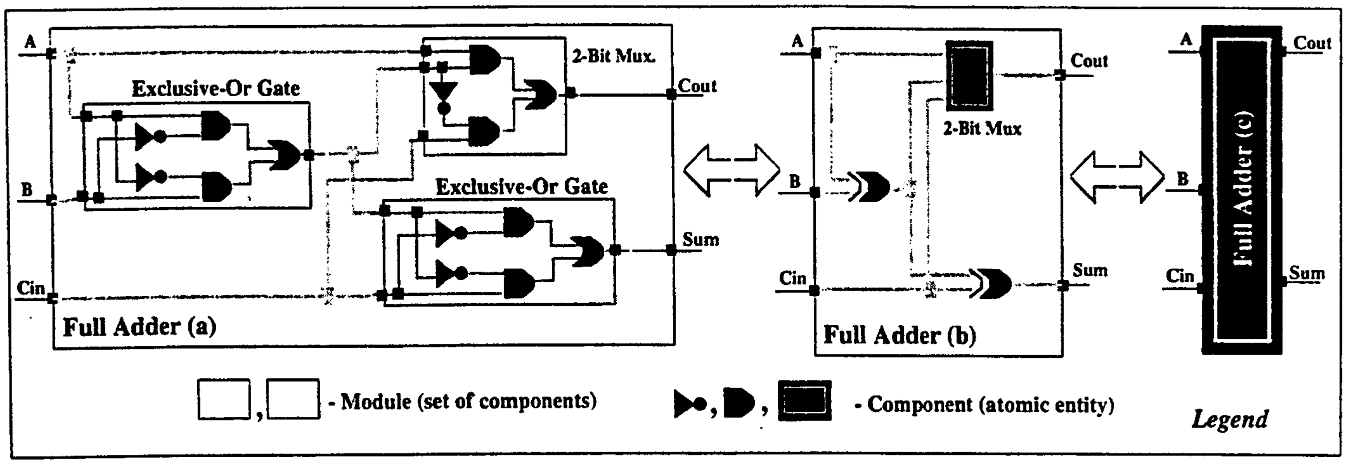


Figure 1: Functionally Equivalent Models for a Full Adder

der to improve the overall simulation time. However, the impact of a DCS is dependent on the model; *i.e.*, a given DCS may improve, deteriorate, or have no impact on the simulation performance. Therefore, it is crucial to identify and utilize ideal combinations (or sequences) of DCS in order to improve the overall efficiency of a simulation study. However, to determine an ideal sequence of transformations, exhaustive analysis of the possible combinations of transforms must be performed. The analysis is further complicated in the case of web-based simulations because they involve several asynchronous, concurrent operations. Exhaustively analyzing combinations of transformations results in combinatorial explosion of the problem space and is not a realistic approach even for medium sized models.

In an endeavor to engineer a more practical approach for identifying and utilizing efficiency improving substitutions, this paper proposes a novel methodology called *DCS prediction methodology* (DCSPPM). DCSPPM identifies efficiency improving DCS transformations using quantitative measures for generated through a combination of: static analysis of the model, empirical measures of the event granularities (time taken to process an event) of components, estimates of the communication latencies between workstations used for parallel simulation, and by applying heuristics to predict synchronization overheads. This paper presents the design, implementation, and testing of DCSPPM in WESE. Section 2 presents an overview of WESE. Section 3 presents a detailed description of DCSPPMalong with the issues involved in the implementing DCSPPM in WESE. Some of the experiments conducted to evaluate the accuracy of the estimates generated by DCSPPM are discussed in Section 4. Section 5 concludes the paper and presents some of the ongoing work.

2 WESE

This section presents only a brief overview of WESEto aid further discussions in the remainder of the paper. A detailed description of WESE and DCS is available in the literature (Rao, Chernyakhovsky, and Wilsey 2000; Rao and Wilsey 2000; Rao, Wilsey, and Carter 2001). WESE provides a component based modeling language, a framework for developing a web-based repository of components, and the infrastructure for distributed simulation. An overview of WESE is shown in Figure 2. WESE provides both an HTML interface and a text based frontend that can be used to interact with the WESE server. The server controls and coordinates the various parallel and distributed activities of the system. The primary input to WESE is the model of the system described using the System Specification Language (SSL). The specification of a model or an SSL design file consists of a set of interconnected *modules*. Each module consists of three main sections, namely: (i) the *component definition section* that contains the details of the components to be used to specify a module (such as the Universal Resource Locator (URL) of a factory and name of the source object along with initial parameters); (ii) the *component instantiation section* that defines the various components constituting the module; and (iii) the *netlist section* that defines the interconnectivity between the various instantiated components. SSL permits an equivalent component to be associated with each module. DCS is performed by replacing the module with its equivalent component or vice versa.

SSL also allows an optional *label* to be associated with each module. The *label* can be used as a component definition in subsequent module specifications to nest one module within another. This technique can be employed to reuse module descriptions and develop hierarchical specifications. As shown in Figure 2, the input SSL source is parsed into an object-oriented (OO) in-memory *intermediate form* (SSL-IF). Hierarchical SSL models are elaborated or "flattened" prior to simulation by the elaborator (Rao, Chernyakhovsky, and Wilsey 2000). Elaboration is a recursive process that flattens a hierarchical model by substituting each module reference (made through the use of *labels*) with an unique instance of the module.

The WESE server also performs the task of collaborating with the distributed factories and coordinating the simulations. The DCSPPM module houses the implementation of the proposed DCSPPM. A detailed description of DCSPPM is presented in Section 3. The *simulation manager* (Figure 2) performs the activities associated with coordinating with the object factories (via the *factory manager*) to setup a distributed simulation. The *factory manager* performs the tasks of interacting with the distributed WESE factories using a predefined protocol. A WESE factory can be viewed as a web-based repository of components with added capability to simulate them. Parallelism occurs at the factory level *i.e.,* each factory is a parallel, asynchronous simulation infrastructure (Rao, Chernyakhovsky, and Wilsey 2000). Parallel simulations are performed by utilizing components (or simulation objects) from different factories. a WESE factory is built from sub-factories and *object stubs*. Object stubs contain attributes of the a component such as interface description, cost, and formal specifications. The *simulation sub-system* of a WESE factory is built around the WARPED simulation kernel. WARPED is an API for a general purpose discrete event simulation kernel with different implementations (Radhakrishnan, Martin, Chetlur, Rao, and Wilsey 1998). WESE utilizes the Time Warp (Radhakrishnan, Martin, Chetlur, Rao, and Wilsey 1998) based simulation kernel of WARPED. It provides the infrastructure for distributed simulation and also performs the task of enabling DCS. A more detailed description of WARPED and Time Warp are available in the literature (Jefferson 1985; Radhakrishnan, Martin, Chetlur, Rao, and Wilsey 1998).

In WESE, an event-driven mechanism has been employed to sequence the various phases involved in DCS. A component can trigger DCS by merely scheduling an appropriate kernel event. WESE also provides a simple API for mapping states of components during DCS.

3 DCS PERFORMANCE PREDICTION METHODOLOGY

The *DCS performance prediction methodology* (DCSPPM) has been developed to ease exploration of different configurations of a model to determine sequences of efficiency improving DCS transformations. Prior to DCSPPM, several techniques were explored to predict the changes in performance induced by DCS (Rao, Wilsey, and Carter 2001). However, the results obtained from these techniques had considerable errors (in the range of ±30% to ±50%), particularly in parallel simulation scenarios. Since the error factors were large, the earlier techniques were practically unusable. Consequently, one of the primary motivations for developing DCSPPM was to design a more accurate methodology.

In DCSPPM, identification of DCS transformations is performed by comparing the empirical estimates generated by DCSPPM for each DCS transformation. The empirical estimates generated by DCSPPM indicate the *changes* induced by a transformation on various model and simulation related parameters such as: modeling costs, observability of the model, and change in simulation performance. The estimates can also be viewed as weights associated with each transformation. Consequently, identifying ideal sequences of transformations can be reduced to an optimization problem of choosing a sequence of transformations such that the sum of their weights is optimal.

For example, consider a scenario that involves three DCS transformations, say t_1, t_2, and t_3 and DCSPPM is used to estimate the changes induced by these transformations. An example of the change in observability generated by DCSPPM would be -10%, -5%, -5% for t_1, t_2, and t_3 respectively. Let the change in simulation times estimated by DCSPPM be $+10\%$, -5%, and -5%. Positive values indicate increase in the quantitative estimate of the given parameter while negative values indicate decrease. Let us assume that the objective is to minimize simulation time with minimal decrease in observability. In this case, an ideal sequence of DCS can be chosen by selecting those transformations which decrease in simulation time is better than the decrease in observability *i.e.,* t_2 and t_3 are the candidates while t_1 is not. In other words, a solution to the given problem would be to use transformations t_2 and t_3 and ignore transformation t_1.

It must be noted that the modeling and simulation costs are independent of each other. However, in practice they must be simultaneously optimized in order to enable ideal tradeoffs. The quantitative estimates generated by DCSPPM are a measure of the *changes* induced by a transform and are not absolute measure. In other words, the goal is to identify the best combination given a set of choices and not the absolute optimal configuration for a given model. In DCSPPM, the changes induced by a transformation on the various parameters are estimated through a combination of: static analysis of the model, empirical measures of the event granularities (time taken to process an event) of components, estimates of the communication latencies between workstations used for parallel simulation, and by applying heuristics to predict synchronization overheads. A detailed description of the techniques used by DCSPPM to generate the quantitative estimate is presented in the following sub-sections.

3.1 Estimation of changes in Modeling Parameters

The model related quantitative estimates generated by DCSPPM are: change in cost of model, changes in observability, and change in level of abstraction are generated

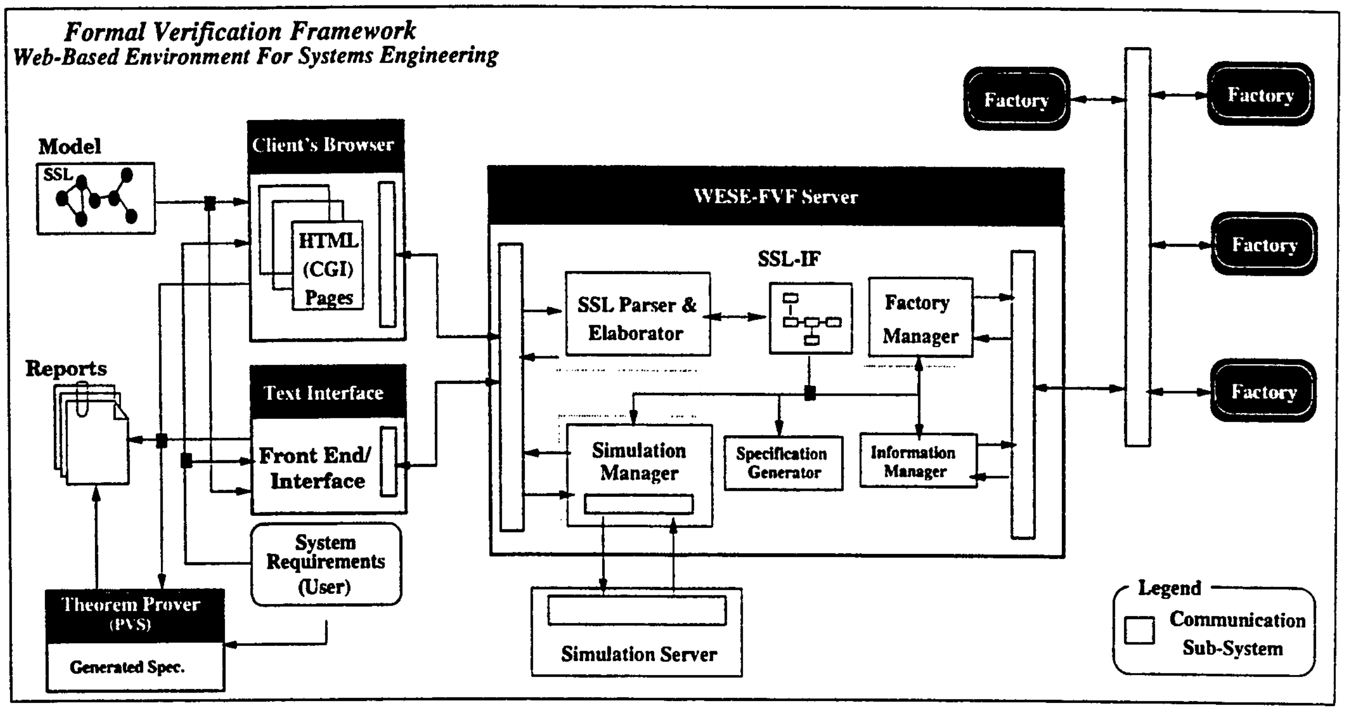


Figure 2: Overview of WESE

based on the components constituting the. These quantitative estimates are dependent solely on the components constituting the model. The changes in modeling costs is an abstract quantity that characterizes the overheads involved in developing or using the components. It is a important measure, particularly in web-based simulations because components obtained by third-party model developers may be used. The third party components may are typically offered as value added services based on different pricing schemes (Rao, Wilsey, and Carter 2001). In DCSPPM the cost of a model is computed as the sum of the costs of the components constituting the model. Different schemes may be used to determine the cost of a component }. In WESE, the number of lines of source code for each component have been used as a measure of cost. The cost of each component is available as a part of the *object stub* associated with each component in a WESE factory (please refer Section 2). The cost of the components is collated by the DCS module present in the WESE server (Figure 2). The changes induced by a transformation to the modeling costs is computed as the percentage change in the overall cost of the model when a module is substituted by its equivalent component (or vice versa).

The change in the total number of ports in the model is used as a measure of the change in observability. In WESE, a *port* is a conceptual point through which some interaction occurs with a component and a set of ports constitute the interface of a component. The number of ports of a component are a part of the SSL description of the model. The total number of ports are computed by summing up the number of ports of each component in the model. When a DCS transformation is applied to a model, the components constituting the model change. Correspondingly, the total number of ports in the model also change. The percentage change in the total number of ports induced by a transform, is reported as a measure of the change in observability. In DCSPPM the change in the level of abstraction is estimated by computing the percentage change in the total number of components and hierarchical levels. The change in observability and level of abstraction is computed by statically analyzing the elaborated SSL description of the model. The object-oriented nature of SSL-IF has been utilized to implement the static analyses. The DCSPPM module, present in the WESE server (Figure 2) handles the task of generating the estimates using the above methodology. The time complexity of this phase of DCSPPM is $O(c)$, where c is the total number of components in the model.

3.2 Estimation of changes in Simulation Performance

The simulation parameters are dependent on the model as well as the hardware platform used for simulation. The primary parameter computed by DCSPPM is the overall change in simulation time when a transformation is applied to a model. The change in simulation time is measured in terms of the change in the granularity of the model. The granularity of a model is in turn determined by the granularity of the components constituting the model, the platform used for simulation, and the configuration of the simulation (such as number of processors used and partitioning of components).

The granularity of a component represents the average time taken by the component to process an event. In DCSPPM, three factors contribute to the granularity of a component; namely (i) average time taken to process an event; (ii) communication costs involved in receiving the event over communication networks (if any); and (iii) synchronization overheads. These parameters have shown to determine the overall time taken to simulate a model (Balakrishnan, Frey, Abu-Ghazaleh, and Wilsey 1997). The techniques used to estimate these three parameters are discussed below.

3.2.1 Event processing Cost

The event processing cost represents the average time taken to execute one event. The cost of processing an event also includes the simulation kernel overheads such as state saving overheads and event scheduling costs. The event processing costs of a component are experimentally determined by setting-up a temporary "test" simulation and monitoring the time taken to execute each event. The granularity estimation is performed by the WESE factory which houses the component. It must be noted that the simulation is also performed by the same factory (or workstation). The WESE factory provides an API that must be used by the component-developer to define the test simulation to be used. The granularities are assumed to follow a Normal distribution in concordance with statistical theories (Hogg and Craig 1995; Jain 1991). Suitable (95%) confidence intervals are also computed and stored in the stubs.

The API also provides support for estimating the event processing costs for components that have multiple, distinct regions — the time taken to process an event significantly varies (based on the modeler's discretion) from event to event. In this case, each distinct region is assumed to follow a Normal distribution (as before) and the overall event processing is defined as a weighted average of each individual distribution. The weights may also be replaced with suitable probability values which indicate the probability with which a given type of event may be received by the component. The resulting weighted average also follows a Normal distribution with a given mean and variance. In WESE, granularities of each unique components is computed once and reused. The worst case time complexity of this phase in DCSPPM is $O(c)$.

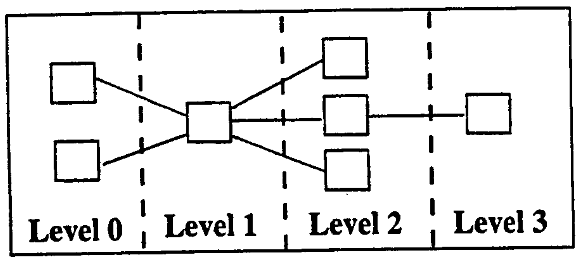
3.2.2 Estimation of Communication Costs

Communication latencies strongly influence the overall time taken for parallel simulations (Balakrishnan, Frey, Abu-Ghazaleh, and Wilsey 1997). In WESE, communication latencies arise when components from two distinct WESE factories are used to develop a module; *i.e.*, the events generated by the components have to be delivered to the target component via communication networks. On the other hand, event exchanges between components on the same factory is performed through simple pointer manipulation (by the WARPED kernel) and the overheads are included as a part of the the event processing costs (as explained above).

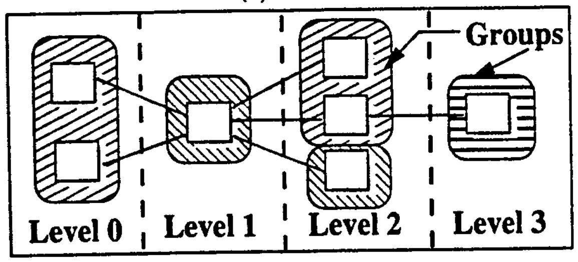
In DCSPPM, the communication latencies between components is estimated using the following 3 steps:

1. Levelization: During the first phase of analysis, the components and modules constituting a model are "levelized" (or ordered) such that the inputs of a component are at a lower level. Figure 3(a) illustrates an example of a levelized model. Levelization captures the flow of "inputs to outputs" in the model (from left to right in Figure 3). In other words, events in the model flow from a lower level to a higher level. The levels represent inherently serial blocks of computations in the model. Any parallelism in the model occurs in between components in each level. Since levelization requires each interconnection to be inspected, the time complexity of this phase is $O(n)$, where n is the total number of netlists (or interconnections) in the model.
2. Grouping: Next, the components at each level are grouped together based on their source factories, as shown in Figure 3(b). That is, all the components in a group reside on a given factory. The groups in each level represent the parallel entities. Note that, processing of events within a group proceeds sequentially (based on the construction of the simulation infrastructure of WESE). The time complexity of the grouping step is $O(c)$.
3. Estimation: During the last phase, the average communication latencies between *groups* of components is estimated. Communication delays arise when events are exchanged between the *groups*; within a group the communication costs are zero (as explained earlier). Grouping of components based on factories eases identifying pairs of between which communication latencies need to be measured. Estimation of communication latencies is performed by the WESE factories and is coordinated by the WESE server. Latencies are estimated by exchanging a number of messages between the two factories, measuring the round trip time for the messages, and computing an average. One WESE factory acts as a server while the other acts as a client. Estimation proceeds in a "lazy" manner; *i.e.*, estimation of latencies between a given pair of WESE factories is performed only once. The worst case time complexity of the estimation phase is $O(c/2)$.

The estimated average communication delays is then added to the overall granularity of the component (*i.e.*, total granularity = event processing costs + communication latency). If a component has multiple sources, then the average of the



(a) Levels



(b) Groups

Figure 3: An example of Levels and Groups

communication delay from each source is used. It must be noted that the average communication delay is also assumed to follow a Normal distribution with a given mean and variance.

3.2.3 Estimation of Synchronization Overheads

WESE is a Time Warp synchronized parallel simulation environment. The *rollbacks* that occur in a Time Warp simulation are a direct measure of the synchronization overheads. In DCSPPM the synchronization overheads are represented as the probability with which a component would be rolled-back during simulation. In a Time Warp simulation, rollbacks imply that some of the events must be reprocessed. Accordingly, the overall granularity of the component is increased by this probability factor to account for synchronization costs —*i.e.*, the average granularity of the component increases (by a given percentage) because of rollbacks ($total\_granularity = total\_granularity + (total\_granularity * rollback\_probability)$).

DCSPPM uses a heuristic to estimate the probability of a rollback. The intuition behind this heuristic is that a component will be rolled-back if, concurrent events (events with same simulation time) arrive at different times (real time). The probability of such occurrences increases as the variance in event arrival times at the inputs of component increases. For example, if all the inputs were being generated by components on the same factory, then the probability of a rollback is almost zero. On the other hand, if the inputs were being generated by components with different total granularities, on different factories, the probability of a rollback increases.

The levelized and grouped model (generated earlier) is also used to estimate synchronization costs. The estimation proceeds from the lowest layer to the highest layer, tracing the "natural" flow of events in the model. At each level, the synchronization costs for each component is computed using the proposed heuristic and the total granularity of the components from earlier levels. The rollback probability of components at a lower level is also taken into consideration in order to account for cascading rollbacks. The results from the static analyses are stored back into the intermediate form for future references. The time complexity of this phase of DCSPPM is $O(c * n)$.

3.3 Identifying Efficiency Improving Sequences of DCS

Having estimated the cost, observability, and average granularity of each component, the overall cost, observability, and average granularity of the complement model is computed. This performed my merely summing up the attributes of each component in the model. Changes induced in these attributes by a DCS transformation (*i.e.,* when a module is replaced by a component) is also computed. The change in attribute value is computed as follows. Let a given DCS transformation substitute a module m containing the set of components $m = \{c_1, c_2, \cdots, c_n\}$ by $c_{EC}(c_{EC} \notin m)$. Then, the change in a given attribute a, represented by $\Delta(a)$, is computed as:

$$\Delta(a) = (\sum_{\forall c_i \in m} c_i.a) - c_{EC}.a$$

where $c_x.a$ represents the quantitative estimate of attribute a for component c_x. Moreover, each DCS transformation also involves additional overheads during simulation. These overheads are estimated in terms of the number of kernel events generated to achieve DCS. In WESE, three kernel events (two `update` events, and one `state-value` event) are scheduled for each port in the module being substituted. An average granularity for each kernel event is estimated by each WESE factory and that average is multiplied three times by the number of ports in the module to obtain the DCS overheads — *i.e.,* $DCSoverhead = 3 * |ports| * (average\ DCS\ cost\ of\ one\ port)$. The number of ports in a module is obtained from SSL-IF. All arithmetic operations are performed using statistical operations defined for Normal distributions (Hogg and Craig 1995). The worst case time complexity of this phase of the algorithm is $O(c)$.

DCSPPM utilizes the above described techniques to generate quantitative estimates of the changes induced by a DCS transformation in modeling costs, observability of the model, and simulation performance. These estimates are then used to identify an ideal sequence of DCS based on the user's requirements (as explained in Section 3). The modeler may also manually choose the sequence of DCS based on the

Table 1: Details of Modules used for Experiments

Model Name	Total Number of				Total DCS Analysis Time (sec)			
	Hierarchies	Modules	Components	Netlists	1 F°	2 F°	3 F°	4 F°
4-Bit-Adder	3	4	30	82	0.73	2.51	5.16	6.48
32-Bit-RCA*	4	64	192	1088	9.10	10.30	12.32	13.25
64-Bit-RCA*	5	128	884	2224	17.82	18.42	19.82	20.36
SM$^+$1	4	200	2000	4375	31.06	30.90	31.47	31.48
SM$^+$1	4	300	3000	6850	46.73	45.62	45.19	44.58

Note: *RCA = Ripple Carry Adder; $^+$SM = Synthetic Model; ^{o}F = Factory

estimates generated by DCSPPM. The overall time complexity of DCSPPM is $O(3.5c + n + cn)$, where c is the total number of components and n is the number of interconnections (or netlist entries) in the model. In general, assuming $c << n, O(DCS\text{PPM}) \approx O(n)$.

3.4 Assumptions underlying DCSPPM

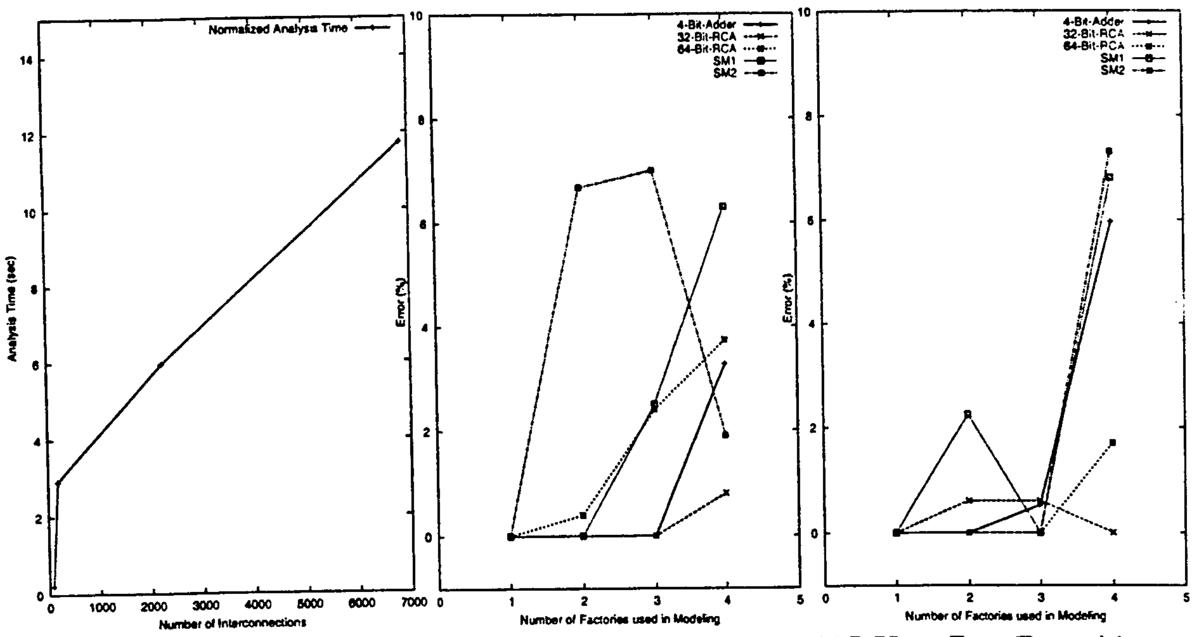
DCSPPM is a static parameter estimation methodology. Several assumptions regarding the model characteristics and the simulation platform have been made during its design and implementation. The assumptions underlying DCSPPM are: (i) the underlying simulation kernel scales linearly with respect to the number of events; (ii) the overheads of enabling DCS is linear with respect to the number of ports in a module; (iii) workload on the workstations does not significantly change during simulation; (iv) communication latencies do not change considerably during simulation; (v) overall granularity of the models does not skew considerably; *i.e.*, the probability with which a component may receive events with different granularities is the same; and (vi) if the model has several different paths from inputs to outputs, then the probability with which each path is taken is equal. The last two assumptions imply that DCSPPM assumes that the behavior of the model (in a given simulation-run) does not deviate significantly from its average behavior. If the behavior is skewed then, in such scenarios the estimates generated by DCSPPM will be inapplicable.

4 EXPERIMENTS

The experiments conducted to evaluate the accuracy of the estimates generated by DCSPPM are presented in this section. The experiments were conducted using a set of digital logic circuits (real world models) and a set of synthetic models. The synthetic models were used to obtain larger benchmarks with a broader range of characteristics and behaviors. They were developed suitably re-targeting the Performance and Scalability Analysis Framework (PSAF) (Balakrishnan, Frey, Abu-Ghazaleh, and Wilsey 1997) backend. PSAF provides a platform-independent Workload Specification Language (WSL) that allows characterization of simulation models using a set of fundamental performance critical parameters. A WESE-specific backend was developed for PSAF in order to obtain the synthetic models. A more detailed description of PSAF along with the API for developing new PSAF-backends is available in the literature (Balakrishnan, Frey, Abu-Ghazaleh, and Wilsey 1997).

Some of the characteristics of the benchmarks used in the experiments are shown in Table 1. These benchmarks were described in SSL by suitably utilizing components from various WESE factories. Larger models were built from smaller sub-modules using the hierarchical model technique supported by SSL. The models also included equivalent component specifications for modules, that get used during DCS. For example, the `4-Bit-Adder` (shown in Table 1) is implemented using a set of `Full Adders`. Each `Full Adder` is specified using a set of basic gates along with auxiliary component specifications (Figure 1). Parallel simulation experiments were conducted by suitably modifying the SSL descriptions to utilize components from a given number of WESE factories. For parallel simulation, the WESE factories are deployed on a network of shared memory multiprocessor (SMP) workstations running Linux. Each workstation consists of two 166MHz Pentium Pro Processors with 128MB of memory. Two factories are deployed per workstation and the workstations are networked using fast Ethernet.

The time taken for analyzing different configurations of the models, using a varying number of WESE factories is shown in Table 1. The number of modules (shown in Table 1) in each benchmark also indicates the number of DCS transformations that had to be analyzed. The timing information shown is the graph is the average of 10 runs. The analysis times shown in Table 1 also include the time taken to estimate the communication latencies between different WESE factories. Figure 4(a) illustrates the average analysis time without communication delays for the different model configurations. The timing in Figure 4(a) has been normalized with respect to the number of interconnections (or netlists) present the models. As shown in Figure 4(a), the



(a) Normalized analysis Time (b) DCSPPM Error (Static) (c) DCSPPM Error (Dynamic)

Figure 4: Results from Empirical Evaluation of DCSPPM

time for analyzing a model varies linearly with respect to the number of interconnections (or edges) in the model. The graph confirms the expected time complexity (Section 3) of DCSPPM to be approximately $O(n)$, where n is the number of interconnections in the model.

The graphs in Figure 4(b) and Figure 4(c) presents the error in the estimates of simulation time generated by DCSPPM for both static and dynamic component substitution cases. The error percentages were computed by comparing the predicted changes in simulation time against the observed changes. The error value indicates the deviation of the observed data from the 95% confidence interval of the corresponding value predicted by DCSPPM. The simulations involving dynamic transformations did not involve any static transformations (and vice versa) in order to clearly distinguish the results obtained in the two cases. In addition, no additional jobs were run on the various workstations used for simulation — *i.e.,* the load on the workstations was almost a constant throughout the experiments. It must be noted that the errors in estimates of model related parameters (such as change in costs and observability) were zero because they are deterministic estimates. In other words, they are generated through static analysis of the model and not using empirical estimates.

As illustrated by the graphs in Figure 4(b) and Figure 4(c), the estimated change in simulation time closely reflects the observed change in simulation time. The maximum error in the estimations was about 8% even though the predicted variance in the estimated values are relatively small ($\pm$2% to $\pm$7%). As illustrated by the experiments, the predicted changes in simulation time (with 95% confidence intervals) closely track the observed changes in simulation time demonstrating the accuracy of the estimation technique used in DCSPPM. The estimates in the case of 1 factory simulations (inherently sequential) are accurate because of absence of non-deterministic factors such as communication latencies and rollbacks. As illustrated by the graphs in Figure 4, the estimates generated by DCSPPM closely track the actual changes that occur during simulation. The experiments highlighting the effectiveness of the estimation methodology used in WESE.

5 CONCLUSIONS

The design and implementation of DCSPPM, a methodology for performance estimation of static and dynamic component substitution, was described in this paper. DCS, coupled with DCSPPM, is is an effective technique to enable more optimal tradeoffs between several model and simulation related parameters. They make WESE a controlled environment for conducting simulations — a model developer can utilize the estimates to intelligently fine tune the simulations to achieve maximum efficiency. DCSPPM has a polynomial time complexity and significantly reduces the overheads involved in exhaustively analyzing all possible combinations of DCS transformations. For example, a straightforward "greedy" algorithm (similar to 0/1 Knapsack algorithm) can be employed to obtain a sequence of transforms that optimize a model for a given combination of the modeling and simulation parameters (an optimizing function along one or more axes). It must be noted that DCSPPM aims to identify the best combination given a set of choices and not the absolute optimal configuration for a given model. The experiments presented in this paper show that the predicted changes closely track (with an error of ±8%) the observed changes, highlighting the effectiveness of DCSPPM. The estimates may also be used as indicators for further model development and refinement. Currently, work is underway to relax some of the assumptions underlying DCSPPM. Studies are also being conducted to adapt DCSPPM for conservatively synchronized parallel simulations. As indicated by our studies, DCSPPMprovides a effective methodology to estimate the changes induced by a sequence of DCS in several modeling and simulation parameters.

REFERENCES

Balakrishnan, V., P. Frey, N. Abu-Ghazaleh, and P. A. Wilsey. 1997, December. A framework for performance analysis of parallel discrete event simulators. In *Proceedings of the 1997 Winter Simulation Conference*. Winter Simulation Conference.

Hogg, R. V., and A. T. Craig. 1995. *Introduction to mathematical statistics*. Englewood Cliffs, New Jersey: Prentice Hall.

Jain, R. 1991, April. *The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling*. New York, NY: Wiley-Interscience.

Jefferson, D. 1985, July. Virtual time. *ACM Transactions on Programming Languages and Systems* 7 (3): 405–425.

Radhakrishnan, R., D. E. Martin, M. Chetlur, D. M. Rao, and P. A. Wilsey. 1998, December. An Object-Oriented Time Warp Simulation Kernel. In *Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE'98)*, ed. D. Caromel, R. R. Oldehoeft, and M. Tholburn, Volume LNCS 1505, 13–23. Springer-Verlag.

Rao, D. M., V. Chernyakhovsky, and P. A. Wilsey. 2000, January. WESE: A Web-based Environment for Systems Engineering. In *2000 International Conference On Web-Based Modelling & Simulation (WebSim'2000)*. Society for Computer Simulation.

Rao, D. M., and P. A. Wilsey. 2000, December. Dynamic component substitution in web-based simulation. In *In Proceedings of the 2000 Winter Simulation Conference (WSC'2000)*. Society for Computer Simulation.

Rao, D. M., P. A. Wilsey, and H. W. Carter. 2001, April. Optimizing costs of web-based modeling and simulation. In *Proceedings of the First International Workshop on Internet Computing and E-Commerce (ICEC'01)*. IPDPS.

BIOGRAPHIES

DHANANJAI M. RAO (`dmadhava@ececs.uc.edu`) is a Ph.D. candidate in the Department of Electrical and Computer Engineering & Computer Science at the University of Cincinnati. He received his Masters from the same department in August 2000. He received his Bachelor's degree in Computer Science and Engineering from the University of Madras, India in 1996. His research interests include parallel discrete event driven simulation, distributed computing, network simulation, object oriented design patterns, and web-based simulation.

PHILIP A. WILSEY (`philip.wilsey@uc.edu`) is an Associate Professor in the Department of Electrical & Computer Engineering and Computer Science at the University of Cincinnati. He received PhD and MS degrees in Computer Science from the University of Louisiana at Lafayette and a BS degree in Mathematics from Illinois State University. His current research interests are parallel and distributed processing, parallel discrete event driven simulation, computer aided design, formal methods and design verification, and computer architecture. He is a senior member of the IEEE and is a member of the IEEE Computer Society and the ACM.

ACKNOWLEDGMENTS

Support for this work was provided in part by the Ohio Board of Regents.

Multi-resolution Network Simulations using Dynamic Component Substitution*

Dhananjai M. Rao and Philip A. Wilsey
Dept. of ECECS, PO Box 210030, Cincinnati, OH 45221–0030.
dmadhava@ececs.uc.edu, philip.wilsey@uc.edu

Abstract

Modeling and simulation of large, high resolution network models is a time consuming task even when parallel simulation techniques are employed. Processing voluminous, detailed simulation data further increases the complexity of analysis. Consequently, the models (or parts of the models) are abstracted to improve performance of the simulations by trading-off model details and fidelity. However, abstraction defeats the purpose of studying high resolution network models and magnifies the problems of validation! An alternative approach is to dynamically *(i.e., during the course of simulation) change the resolution of the model (or parts of the model). In our component based Network Modeling and Simulation Framework (NMSF), we have enabled dynamic changes to the resolution of a model using a novel methodology called* Dynamic Component Substitution *(DCS). Using DCS, a set of components can be substituted by a functionally equivalent component (or vice versa) to change the resolution (or the level of abstraction) of a network model. DCS improves the overall efficiency of simulations through dynamic tradeoffs between resolution of a model, simulation performance, and analysis overheads. This paper presents an overview of DCS and the issues involved in enabling DCS in NMSF, an optimistically synchronized parallel simulation framework. The experiments conducted to evaluate the effectiveness of DCS are also illustrated. Our studies indicate that DCS provides an effective technique to considerably improve the overall efficiency of network simulations.*

1 Introduction

Computer and communication networks have steadily grown in size and complexity to meet the growing needs and demands of modern computing [7]. Today's networks involve complex interactions between a few thousand to several million networking components. Study and analysis of modern networks is usually performed using computer based simulations. Parallel simulation techniques are often employed to enable simulation of large network models in acceptable time frames [3]. In simulation studies, the validity of the models plays a crucial role [9]. The models should reflect the size and complexity of the system in order to ensure that crucial scalability issues do not dominate during validation of simulation results [7, 10]. High resolution models are important for studying events that are rare — events that do not even occur in small, low fidelity models may be common in the actual network [7]. Detailed simulation of the complete network is necessary to study large scale characteristics, long term phenomena, and to analyze the network as a whole [9].

However, simulation of high fidelity, high resolution models of large networks is a time consuming task even when parallel simulation techniques are exploited [9]. Furthermore, study of large networks is typically conducted in phases [3], wherein each phase focuses on a particular aspect or portion of the network. In such studies, detailed simulation data from other parts of the network model is inconsequential (depending on the analysis requirements) [3, 10]. Processing voluminous, inconsequential simulation data further aggravates the problems associated with modeling, simulation and analysis. Consequently, parts of the network model that are not critical for a given study are suitably abstracted to minimize inconsequential simulation results and to improve simulation time [3]. Abstraction of selected parts of a model is an effective technique to optimize the overheads associated with modeling, simulation, and analysis by trading-off resolution or details. However, abstraction results in low resolution and possibly low fidelity network models [3, 10] which not only defeats the goal of studying detailed models but also brings us back to the problems of validity of the simulation!

An alternative approach to improve the overall efficiency of large, high resolution network simulations is to *dynamically* (*i.e.,* during the course of simulation) change the resolution (or "level of abstraction") of selected parts of a network model. In this methodology, the resolution of the model is dynamically altered to suit the needs of the

*Support for this work was provided in part by the Ohio Board of Regents.

simulation study – scenarios of interest (or parts of the model) can be simulated in high resolution while the remainder of the simulation (or the model) can be simulated in low resolution. In our component based network modeling and parallel simulation framework, we have enabled dynamic changes to the resolution of a model through a novel methodology called *Dynamic Component Substitution* (DCS). Using DCS, a *set* of components (constituting a network model) can be substituted with *a* (functionally) equivalent component (or vice versa) during simulation, to achieve dynamic changes to the resolution of a model. DCS improves the overall efficiency of a simulation by enabling dynamic tradeoffs between several parameters such as: cost of modeling, resolution (or details) of simulation data, and simulation overheads.

This paper presents an overview of DCS along with the issues involved in enabling DCS in an optimistically synchronized (based on Time Warp) parallel network simulation framework. Section 2 presents an overview of some of the closely related research activities. An overview of DCS along with a brief description of the component based modeling methodology used in this study are presented in Section 3. To ease study and effective use of DCS, an existing Network Modeling and Simulation Framework (NMSF) [9] has been extended to provide support for DCS. An overview of the NMSF is presented in Section 4. The issues involved in extending NMSF to support DCS are discussed in Section 5. The results obtained from some of the experiments conducted to evaluate the overall effectiveness of DCS are presented in Section 6. Section 7 concludes the paper and presents some pointers to future work.

2 Related Research

A number of studies have been reported on selectively abstracting parts of a model to enable efficient tradeoffs between several model and simulation related parameters, such as: model resolution, fidelity, and simulation performance. In this section we present a brief overview of some of the closely related research activities. One of the recent studies on selective abstraction was presented by Huang *et al* [3]. In their work, they present two abstraction techniques for abstracting network and transport layer protocols. They apply the abstraction techniques to the simulation study of reliable multicast protocols. Their studies indicate that, although the abstract simulations are not identical to more detailed simulations, the abstract models provide good approximations and considerably improve simulation performance [3]. Ahn *et al* [1] demonstrate that abstraction can be employed to adjust the simulation granularity of packet network models in order to efficiently study flow and congestion control algorithms. Hybrid simulation models, wherein the network model is a combination of discrete-event and analytical components, have been used to yield efficient, yet accurate simulations [11]. Selective abstraction of models have also been applied to other domains of simulations, particularly for digital logic simulations. Levelized code compilation techniques, that selectively replace parts of combinatorial logic circuits with equivalent behavioral descriptions, are widely used to improve performance of circuit simulations [12]. McBrayer *et al* have described techniques for combining processes descriptions specified in VHDL to yield more abstract models, in order to improve performance of parallel logic simulations [5]. These techniques are all *static*, *i.e.,* selective abstraction of the model is done prior to commencement of the simulation; while, in this study we propose to dynamically change the resolution of selected parts of the model. The use of mixed resolution models to enable effective simulation and analysis of large systems have also been studied [2, 6]. Natrajan *et al* [6] present the issues involved in the use of Multiple Resolution Entities (MRE) in parallel simulations. MREs are entities that are capable of maintaining internal consistency across multiple, concurrent levels of resolution. In other words, a MRE is a model of a sub-system, that is capable of modeling the behavior of the sub-system at different levels of resolution. On the other hand, we propose to dynamically substitute a given set of components with an equivalent component (or vice versa) to change the resolution (and consequently the level of abstraction) of a model.

3 Dynamic Component Substitution

Component based modeling techniques are widely used because they offer a number of advantages [9]. In a component based model, a system is represented as a set of interconnected *components*. A component is a well defined entity which is viewed as a "black box", *i.e.,* only its interface is of interest and not its implementation. A component could in turn be specified using a set of sub-components. During simulation, each *atomic* component is associated with a specific, well defined software module called a *simulation object* or a *logical process* (LP) that implements its behavior and functionality. In such models, a set of components can be substituted by a functionally equivalent component (or vise versa), without altering the basic characteristics of the model. For example, consider the network model shown in Figure 1(a), consisting of 9 nodes interconnected by 3 switches (labeled $1 \cdots 12$); logically organized as 3 sub-networks. Figure 1(b) illustrates a typical component based simulation layout for the network model shown in Figure 1(a). The interconnection between the networking entities (shown as black lines in Figure 1(a)) are represented using logical links (shown as gray lines in Figure 1(b) and Figure 1(c)); *i.e.,* the logical links indicate the pattern in which the components communicate (*i.e.,* exchange events)

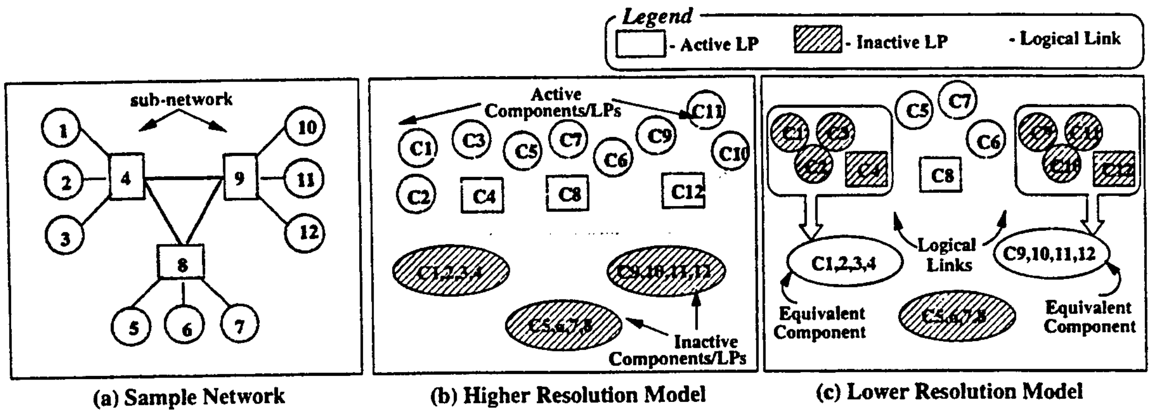


Figure 1. A sample network, typical component based model, and changes induced by DCS

during simulation.

Figure 1(b) also illustrates equivalent components (labeled $C_{1,2,3,4}$, $C_{5,6,7,8}$, and $C_{9,10,11,12}$) for the three sub-networks constituting the model. An equivalent component is capable of closely imitating (within acceptable error margins) the behavior and functionality of the set of components it represents. For example, as shown in Figure 1(c), the component $C_{1,2,3,4}$ is equivalent in behavior and functionality to the set of components $\{C_1, C_2, C_3, C_4\}$. Figure 1(c) illustrates a scenario that could arise when a given set of components are substituted by their equivalent component. The set of components $\{C_1, C_2, C_3, C_4\}$ and $\{C_9, C_{10}, C_{11}, C_{12}\}$ have been substituted by the corresponding equivalent components. As illustrated by Figure 1(b) and Figure 1(c), substitution of components involves updating the logical links between the components and possibly updating the states of the newly activated (or created) components to reflect the current state of the simulation. The simulation configuration in Figure 1(b) is a higher resolution equivalent of the simulation configuration shown in Figure 1(c). It must be noted that the comparison of resolution (or level of abstraction) is only relative and not absolute.

Substitution of components may be done *statically* (*i.e.*, prior to simulation) or *dynamically i.e.*, during the course of simulation [9]. Static component substitution is widely used in different flavors, to address capacity and performance issues of large scale simulations [3, 5, 12]. They are also prevalently used for "What-if" type of simulation analysis. The primary drawback of static component substitution is that functionality, observability, and model details cannot be altered during simulation. However, resolution and fidelity are crucial for effectively studying large scale networks. On the other hand, substituting components during simulation provides a dynamic tradeoff between model details and performance of the simulation. Dynamic Component Substitution (DCS) not only encompasses the utility of its static counterpart but also provides a number of other useful features [9]. It can be used to dynamically change the resolution of selected parts of a model; thereby optimizing simulation overheads and volume of inconsequential simulation data. DCS provides an effective solution for improving the overall efficiency of large scale network simulation and analysis. The improvement in efficiency is achieved by striking a tradeoff between several modeling and simulation related parameters [9]. However, enabling support for DCS in a parallel simulation environment involves additional overheads during kernel and model development. The issues involved in the design and development of a parallel simulation framework capable of supporting DCS is presented in the following sections.

4 Network Modeling and Simulation Framework (NMSF)

A simulation framework capable of supporting DCS has been developed by suitably extending an existing Network Modeling and Simulation Framework (NMSF) [9]. Although, a detailed description of the framework is available in the literature [9], a brief description of NMSF is presented in this section for completeness of this paper and for the discussion (in Section 5) on extending NMSF to support DCS. Figure 2 presents an over view of NMSF. As shown in Figure 2, the primary input to the framework is the model (or topology) of the network to be simulated. The topology of the model is described using the Topology Specification Language (TSL) [9]. TSL is a component based, modular network topology modeling language wherein a network is specified as a set of interconnected networking components (such as traffic generators, nodes, and routers). The components are developed using the Application Program Interface (API) provided by NMSF (as shown in Fig-

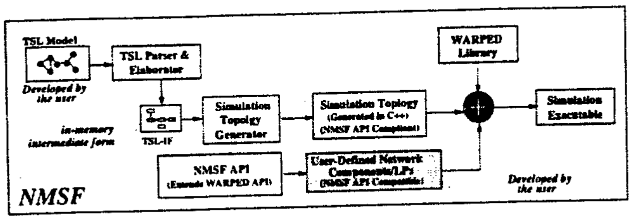

TSL Model
Developed by the user
TSL Parser & Elaborator
in-memory intermediate form
TSL-IF
Simulation Topology Generator
Simulation Topology (Generated in C++) (NMSF API Compliant)
WARPED Library
Simulation Executable
NMSF API (Extends WARPED API)
User-Defined Network Components/LPs (NMSF API Compatible)
Developed by the user
NMSF

Figure 2. Overview of NMSF

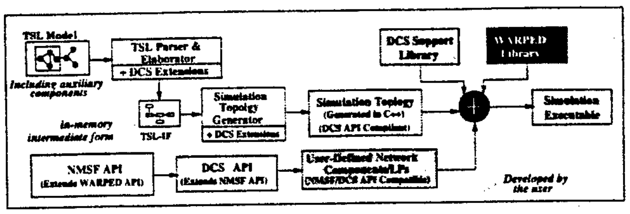

TSL Model
Including auxiliary components
TSL Parser & Elaborator
+ DCS Extensions
in-memory intermediate form
TSL-IF
Simulation Topology Generator
+ DCS Extensions
Simulation Topology (Generated in C++) (DCS API Compliant)
DCS Support Library
WARPED Library
Simulation Executable
NMSF API (Extends WARPED API)
DCS API (Extends NMSF API)
User-Defined Network Components/LPs (NMSF/DCS API Compatible)
Developed by the user

Figure 3. NMSF with DCS Extensions

ure 2). As shown in Figure 2, the input TSL description is parsed into an object-oriented in-memory intermediate form called TSL-IF. TSL-IF is used by a backend code-generator to generate a simulatable network topology. The generated code contains the necessary calls to instantiate the various components (developed using the NMSF API), pass parameters specified in the TSL description to them, configure, and establish the simulation. The generated code, in conjunction with all the other components of NMSF, is in C++. The generated code is compiled along with the WARPED [8] (a parallel simulation kernel) library and the user-defined modules to obtain the final executable; which when run performs the actual simulation. A more detailed description of the different components relevant to this study are presented in the following subsections, while detailed descriptions of the other components constituting NMSF are available in the literature [9].

4.1 Topology Specification Language (TSL)

TSL provides a hierarchical, component based modeling techniques for specifying the topology of a network for simulation. A TSL specification consists of a set of interconnected sub-topology specifications. Each sub-topology specification consists of three main sections, namely the *object definition section*, the *object instantiation section*, and the *netlist section*. The object definition section contains the details of the components (such as the name of the C++ class used to model the component along with necessary parameters) used in the sub-topology. The object instantiation section specifies the set of components constituting the sub-topology. Each object instantiation must be associated with an object definition. The object definitions and instantiations are synonymous to defining a new type and variables of a given type in a programming language. The netlist section defines the interconnectivity (or communication pattern) between the component instantiations. An optional *label* may be associated with each sub-topology. The labels may be used as an object definition in subsequent sub-topology specifications to nest one sub-topology within another. In other words, a sub-topology encapsulates a set of interconnected components and provides a predefined interface to them. When sub-topologies are nested within one another, they get interconnected through the predefined interface to form larger networks. Hierarchical topologies are elaborated or "flattened" prior to simulation [9]. An example TSL source code is presented in Section 5 while further details on TSL is available in the literature [9].

4.2 WARPED

The parallel simulation capabilities of NMSF have been enabled using WARPED [8]. WARPED is an API for a general purpose discrete event simulation kernel with different implementations [8]. NMSF utilizes the Time Warp synchronized simulation kernel of WARPED for parallel simulation. A Time Warp synchronized simulation is organized as a set of communicating asynchronous logical processes (LPs). The LPs operate as asynchronous discrete event simulators and communicate between each other by exchanging *virtual time*-stamped event messages [4]. Virtual Time is used to model the passage of time and defines order on the events in the system. Accordingly, the WARPED kernel [8] provides an API to build different LPs with unique definitions of state. It provides the basic functionality for sending and receiving events between LPs. Control is exchanged between the application and the kernel through cooperative use of function calls. In WARPED, LPs are placed into groups called "clusters" to optimize communication overheads. Although, LPs are grouped together into clusters they are not coerced into synchronizing with each other. Causal violations are detected by a LP when it receives an event with time-stamp lower than its local time (LVT). Such events are called *straggler* events. On receiving a straggler event, a *rollback* mechanism [4] is invoked to recover from the causality error. The WARPED kernel insulates the application from the intricacies of rollbacks, optimistic synchronization, and other simulation overheads [8].

5 Implementing Support for DCS in NMSF

A simulation environment capable of supporting DCS was developed by suitably extending the NMSF. Care has been taken to ensure that the extensions do not invalidate any existing models for the NMSF. Figure 3 presents an overview of NMSF along with the modifications for supporting DCS. As shown in Figure 3, the first phase of im-

```
subNet : abstSubNet "nodes=3 trGens=3" {
    node : simpleNode "CBR size=50";
    router : Router;
    trGen : Generator "const delay=1ms pkts=50"; }
{ n1, n2, n3 : Node;
    tr1, tr2, tr3 : trGen;
    r1 : router; }
{ t1 : n1; t2 : n2; t3 : n3;
    n1 : r1; n2 : r2; n3 : r3;
    r1 : n1 n2 n3 subNet; }
```

```
mainNet {
    controller : DCSController;
    router : Router; }
{ ctrl1 : controller "10ns trigger";
    ctrl2 : controller "20ns trigger";
    subNet1, subNet2,
        subNet3 : subNet;
    r2 : router; }
{ ctrl1 : $subNet1;
    ctrl2 : $subNet2;
    r2 : subNet1 subNet2 subNet3;
}
```

Figure 4. Sample TSL source

plementing support for DCS in NMSF involved extensions to the modeling infrastructure. Changes to the modeling infrastructure involved extending TSL and correspondingly modifying the parser, elaborator, and the backend code-generator. TSL was extended to include additional constructs for associating a set of components with an equivalent component. This was achieved by permitting an *auxiliary* object definition to be associated with each sub-topology in a TSL description. On encountering an auxiliary object definition an auxiliary object instantiation is implicitly created (in TSL-IF). The auxiliary component represents the higher level abstraction (or lower resolution model) of the set of components contained in the sub-topology. In other words, when DCS is triggered for a sub-topology, the set of components encapsulated by that sub-topology are substituted using by the auxiliary component or vice versa, as the case maybe.

The TSL source code for the network model shown in Figure 1(a) is shown in Figure 4. The source code also includes auxiliary component specifications and references (shown in bold, in Figure 4) for the sub-topologies constituting the network model. As shown in Figure 4, each `subNet` is associated with an auxiliary component – an abstract sub network (`abstSubNet`). The main network (`mainNet`) consists of three instances of `subNet`. It also includes specifications for controller components that are used to trigger DCS during simulation. It must be noted that, the controllers are not special components but just regular components (developed using NMSF API and an user can modify them to suit the needs) that are geared to perform specific tasks. Each controller is interconnected to the corresponding auxiliary components using suitable netlist entities (such as "`ctrl1: $subNet1;`").

TSL-IF was also extended to correspondingly reflect the changes to the grammar. As shown in Figure 3, the elaborator was also modified to account for the auxiliary components. The elaborator also creates unique instances of the auxiliary components for each unique occurrence (or usage) of a sub-topology. The auxiliary components are an integral part of the elaborated TSL-IF and are identified using special flags in the various data structures. The elaborator was also extended to identify components that form the interface for a sub-topology. For example, the component `r1` shown in Figure 4 is the primary interface for the sub-topology `subNet`. In other words, any interaction with a sub-topology occurs through the primary interface components. It must be noted that several components could constitute the primary interface. This information is utilized during simulation to optimally update netlist entries (or communication links) during DCS (as explained in detail further below). The elaborator also collates information on the set of components contained by each sub-topology. The data collated by the elaborator is embedded into the generated code. The data is utilized during simulation (by the DCS support library modules shown in Figure 3) to achieve DCS.

The second phase of implementing support for DCS involved extending the simulation infrastructure of NMSF. As illustrated in Figure 3, the API supported by NMSF was extended to include necessary support structures for DCS. The DCS API extends the NMSF API such that the modifications do not invalidate existing models and eases use of DCS by insulating the models from the intricacies of enabling DCS. The API classes also provide necessary interfaces for the DCS support library modules that perform the actual task of achieving DCS. As shown in Figure 3, a support library that provides the necessary simulation-time support for enabling DCS has also been developed. These modules not only implement several interface methods of the DCS API but also perform the actual tasks of achieving DCS. In NMSF, an event driven approach has been adopted for sequencing the different stages involved in achieving DCS. The event driven mechanism was utilized because it offers several advantages. The primary advantage is that, it exploits the inherent simulation capabilities of the underlying kernel; thereby abstracting away the intricacies of parallel simulation. On the other hand, the drawbacks of this design are: (i) it introduces additional events (to achieve DCS) during simulation; and (ii) it adds to the state saving overheads in a Time Warp simulation. However, several Time Warp related optimizations can be exploited to minimize these overheads.

A typical sequence of operations performed to achieve DCS are shown in Figure 5. The figure also illustrates the corresponding sequence of transformations that occur to the model during the different phases. As shown in Figure 5, the initial phase involves triggering DCS in the simulation by scheduling an `Activate` or a `DeActivate` event, as the case may be, to the corresponding auxiliary component(s). On receiving an `Activate` or a `DeActivate` event, the auxiliary component schedules corresponding events to the set of LPs (or components) that it is going to substitute. The information on the set of components to be replaced is collated during elaboration and is passed onto the DCS modules (as explained earlier). The necessary state data is also passed on by the deactivating component(s) to

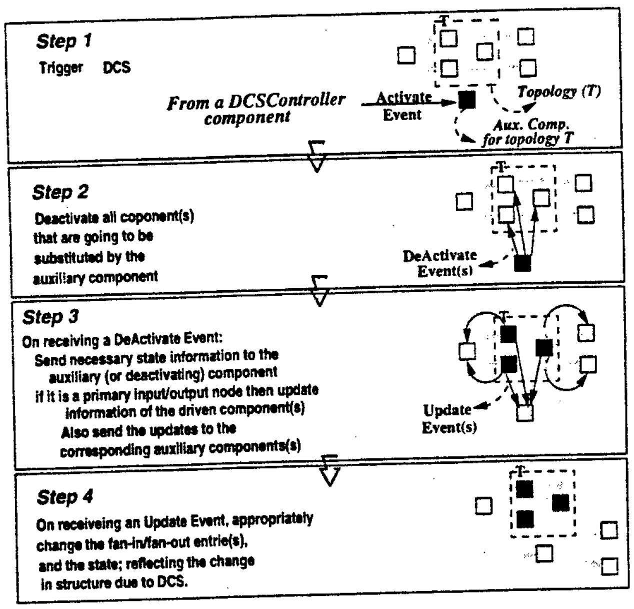

Step 1
Trigger DCS
From a DCSController component
Activate Event
Topology (T)
Aux. Comp. for topology T
T
Step 2
Deactivate all coponent(s) that are going to be substituted by the auxiliary component
DeActivate Event(s)
T
Step 3
On receiving a DeActivate Event:
Send necessary state information to the auxiliary (or deactivating) component
if it is a primary input/output node then update information of the driven component(s)
Also send the updates to the corresponding auxiliary components(s)
Update Event(s)
T
Step 4
On receiveing an Update Event, appropriately change the fan-in/fan-out entrie(s), and the state; reflecting the change in structure due to DCS.
T

Figure 5. Sequence of operations during DCS

the activating component(s). A simple API has been developed for enabling the transfer of states from one component to another. It is the modeler's responsibility to suitably map the states of the various components. In the next phase, necessary `Update` events are scheduled. During the last phase, the `Update` events are processed wherein the LPs update their netlists and states, reflecting the change in structure. As shown in Figure 5, during subsequent simulation cycles, the events generated would be passed on to the new components while the old components get deactivated. It must be noted that the transient events that were already scheduled for the old set of LPs do not get reassigned to the new set of components. They continue to get processed by the substituted (or deactivated) set of components.

6 Experiments

The experiments conducted to evaluate the support for DCS were performed using a set of network models. The network models were built using the hierarchical modeling constructs provided by TSL. The models consisted of a set of interconnected sub-networks. Each sub-network contained a set of nodes connected to a router. The nodes were driven by a set of traffic generators. In the experiments the nodes and the traffic generators were configured to yield a Constant Bit Rate (CBR) type of network traffic with a packet size of 500 bytes. The router component used in the experiments behaves as a typical router that stores and forwards the packets generated by the nodes to the corresponding destination nodes. Interconnectivity between routers is

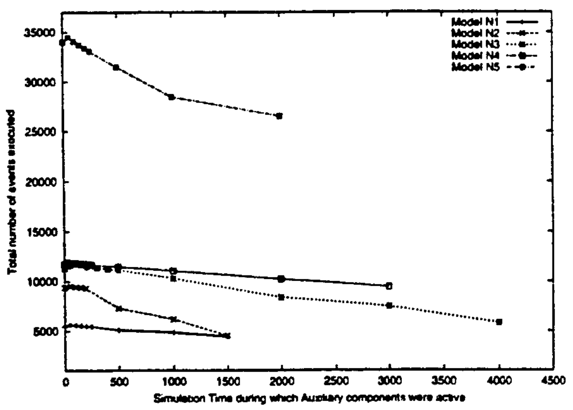
Model Name	Number of Sub Nets.	Number of Components per subNet		
		Nodes	Traffic Gens.	Routers
N1	3	3	3	1
N2	10	5	5	1
N3	5	3	3	1
N4	50	5	5	1
N5	100	6	6	2

Table 1. Models used in experiments

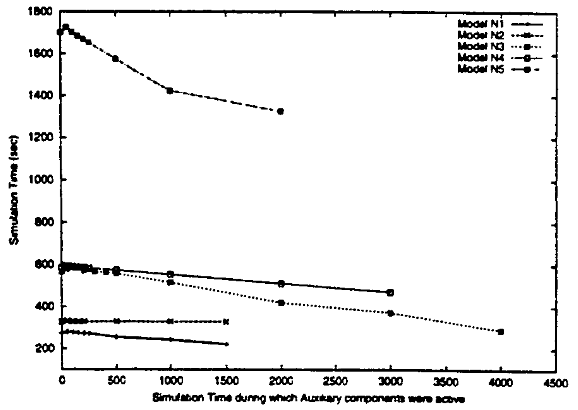
specified using suitable netlist entries in the TSL description of the network model. The routers build necessary routing tables when the simulation is initialized (*i.e.*, just before simulation commences) by exchanging information regarding the set of nodes connected to them. A more detailed description of these components is available in the literature [9].

An abstract sub-topology component that is capable of reflecting the behavior of a sub-network built using the router, node, and traffic generator components was also developed. This component is essentially a router that is also capable of generating network traffic similar to the traffic generated by the nodes in the sub-topology. For example, if a sub-network contained 3 nodes, each node generating traffic at a CBR of 500 bytes/ms, the abstract sub-topology component would *internally* generate 3 packet streams at a CBR of 500 bytes/ms. The necessary information is provided when DCS is triggered (as illustrated in Section 5). The set of events generated by the abstract (or lower resolution) component and the detailed sub-network are equivalent. In other words, the abstract sub-topology component is an equivalent lower resolution model of a sub-topology. An example of a TSL source utilizing these components is shown in Figure 4.

The above described networking components were used to develop different network models by specifying different topologies using TSL. The salient characteristics of the models using in the experiments is shown in Table 1. The TSL descriptions also included equivalent abstract sub-network component (or auxiliary component) specifications for the different sub-topologies. The number of components (*i.e.*, nodes, traffic generators, and routers) substituted by each auxiliary components is shown in Table 1 (column `Number of Components per subNet`). All the simulation experiments were conducted using a network of shared memory multi-processor (SMP) workstations running Linux. Each workstation consisted of two 300MHz Pentium II processors with 128MB of main memory. The workstations were inter-connected using fast Ethernet. Validity of the models and the simulations were verified by comparing the event traces obtained from the top most hierarchical level of each network model. Since, the top level did not involve any DCS, the event traces at that level remain the same, immaterial of the resolution of the sub-networks.



(a) Number of events versus duration of DCS



(b) Simulation Time versus duration of DCS

Figure 6. Effect of duration of DCS on number of events and simulation time

The graph in Figure 6(a) presents the change in the total number of events processed with respect to the duration of simulation time in which the auxiliary components (or lower resolution models) were active. These statistics were collated from the experiments conducted using a single processor where in no rollbacks occur. The data points shown with zero durations did not involve the use of lower resolution modules and represent the basic number of events executed by each model. As illustrated by the graphs shown in Figure 6(a), for short durations during which the auxiliary components are active, the total number of events processed is higher. The increase in number of events is due to the additional events used during DCS. However, as the duration during which the lower level abstractions are active increases, the number of events processed decreases (as shown in Figure 6(a)). The number of events decreases because a set of components are replaced by a single component which results in the elimination of a number intermediate events. Figure 6(b) presents the corresponding simulation times for the different models. As shown by the graphs in Figure 6, the simulation time is proportional to the number of events. However, the change in simulation time for smaller models is not very pronounced because of the low event granularities. As shown in Figure 6(b), the gains in simulation time accrued by utilizing the lower resolution component is significant for the larger models. As illustrated by the graphs in Figure 6, the duration of simulation time for which DCS reduces the number of events varies with respect to the model characteristics. If DCS is triggered at a rate faster than this value the overall simulation time would increase and vice versa. Therefore, this threshold value plays a crucial role in the overall effectiveness of DCS to improve performance of the simulations.

Figure 7 presents the time for simulating model N5 in parallel using a varying number of processors. The LPs were randomly partitioned across the different processors used for simulation. The timing information shown in the graph is the average of 10 simulation runs. As illustrated by the graph, the performance of the simulations increases as the duration during which the auxiliary components are active increase. The improvement in performance is due to the decrease in the total number of events that need to be processed (as shown in Figure 6(a)). As illustrated by Figure 7, the parallel simulations performed using 3 processors performs better than those performed using a single processor. The performance improves because the simulation overheads get distributed across the three processors. In the two processor case the computational overheads dominate the gains accrued by parallel simulation. On the other hand, in the 4 processors case, communication overheads dominate the performance gains. Hence, in these cases the overheads outweigh the gains accrued by employing parallel simulation and the performance of the simulations do not improve. As illustrated by Figure 6 and Figure 7 the overall efficiency of network simulations can be effectively improved using dynamic component substitution.

7 Conclusions

Dynamic change to the level of abstraction of a model (*i.e.*, during simulation) enables more optimal tradeoffs between the resolution (or observability) of a model, model details, and simulation performance. Dynamic Component Substitution is a novel methodology for achieving abstraction of selective parts of a component based model during simulation. In this study, DCS has been applied to improve the overall efficiency of network simulations. The paper presented the issues involved in the design and implementation of the support for DCS in an existing network modeling and simulation framework. A similar approach can be

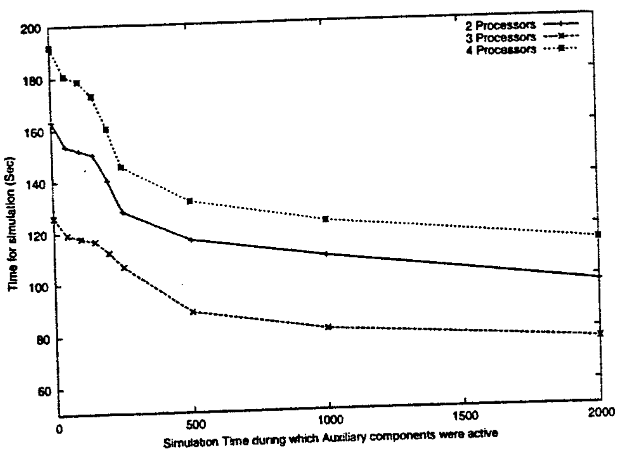


Figure 7. Time for Parallel Simulation

adopted for implementing support for DCS in other network simulators. The experiments in which DCS was used to dynamically change the level of abstraction of the model were described. As illustrated by the results obtained from the experiments, DCS may not always improve simulation performance (which may not be the goal of DCS). For instance, if DCS is triggered very frequently, the overall performance of the simulations may deteriorate. Moreover, reduction in the level of abstraction may not necessarily improve performance of the simulations, parallel simulations in particular. Furthermore, changes in the levels of resolution may introduce errors into the simulation data. Hence, care must be taken while applying DCS. The use of DCS also involves the development of valid components at different levels of abstraction. Albeit some of the modeling overheads, our studies coupled with the experiments presented in this paper highlight that considerable improvements in the overall efficiency of networks simulations can be accrued by employing DCS.

References

[1] AHN, J., AND DANZIG, P. B. Speedup vs. simulation granularity. *IEEE/ACM Transactions on Networking 4*, 5 (Oct. 1996), 743–757.

[2] DAVIS, P. K., AND HILLESTAD, R. J. Families of models that cross levels of resolution: Issues for design , calibration and management. In *Proceedings of the 1993 Winter Simulation Conference* (1993), ACM.

[3] HUANG, P., ESTRIN, D., AND HEIDEMANN, J. Enabling large-scale simulations: Selective abstraction approach to the study of multicast protocols. In *In Proceedings of International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Networks* (Oct. 1998).

[4] JEFFERSON, D. Virtual time. *ACM Transactions on Programming Languages and Systems 7*, 3 (July 1985), 405–425.

[5] MCBRAYER, T., AND WILSEY, P. A. Process combination to increase event granularity in parallel logic simulation. In *9th International Parallel Processing Symposium* (Apr. 1995), pp. 572–578.

[6] NATRAJAN, A., REYNOLDS, P. F., AND SRINIVASAN, S. MRE: A flexible approach to multiresolution modeling. In *Proceedings of the 11th Workshop on Parallel and Distributed Simu lation (PADS'97)* (June 1997), pp. 156–163.

[7] PAXSON, V., AND FLOYD, S. Why we don't know how to simulate the internet. In *In Proceedings of the 1997 Winter Simulation Conference (WSC'97)* (Dec. 1997), pp. 44–50.

[8] RADHAKRISHNAN, R., MARTIN, D. E., CHETLUR, M., RAO, D. M., AND WILSEY, P. A. An Object-Oriented Time Warp Simulation Kernel. In *Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE'98)*, D. Caromel, R. R. Oldehoeft, and M. Tholburn, Eds., vol. LNCS 1505. Springer-Verlag, Dec. 1998, pp. 13–23.

[9] RAO, D. M. A network simulation tool kit. Master's thesis, University of Cincinnati, Aug. 2000.

[10] ROBINSON, S. Simulation model verficiation and validation: Increasing the users' confidence. In *Proceedings of the 1997 Winter Simulation Conference* (Dec. 1997).

[11] SCHWETMAN, H. D. Hybrid simulation models of computer systems. *Communications of the ACM 21*, 9 (1979), 718–723.

[12] WANG, Z., AND MAURER, P. M. Lecsim: A levelized event driven compiled logic simulation. In *Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC'90)* (June 1990), pp. 491–496.

TASK 20

Collaborative Effort in Robust System-on-Chip Design and Simulation

Principal Investigator:	Profs. Carla Purdy and Fred Beyette
Institution:	University of Cincinnati, Cincinnati, Ohio
Period of Performance:	June 20, 1999 to June 10, 2002
Task:	Establish state-of-the-art research modeling, simulation & design activities with the Information Directorate and DAGSI schools in the area of distributed simulation of mixed-technology.

Integrate new projects into the CERC and II Consortium activities for future DARPA funding and workshop symposiums.

Enhance current Air Force modeling and analysis techniques for C4ISR systems.

Teaching Modules for a Class in Mechatronics[1]

FRED R. BEYETTE, JR. AND CARLA N. PURDY
ECECS Department, University of Cincinnati, Cincinnati, OH 45221-0030

Key words: MEMS, mechatronics, rapid prototyping, VHDL-AMS

Abstract: In a previous paper ([3]) we described tools we are using to introduce mechatronic system design to students at the University of Cincinnati. Here we describe the current state of this project. Two main goals for this academic year have been:

- to have students carry out a prototype process to design, fabricate, and test some simple mechatronic devices using the L-EDIT tools, the MOSIS fabrication facilities, and some simple in-house post-processing;
- to develop VHDL-AMS models of devices from the mechanical, fluidic, and temperature domains so students can simulate mixed-domain systems.

Our eventual goal is to integrate our tools and models into a mechatronics design course for advanced undergraduate and beginning graduate students.

[1] We will use the terms *mechatronics* and *MEMS (microelectromechanical systems)* interchangeably to describe systems consisting of components interacting in multiple energy domains.

1. INTRODUCTION

The ECECS Department at the University of Cincinnati offers undergraduate and graduate degrees in electrical engineering, systems engineering, computer engineering, and computer science. Established research programs include digital and analog simulation and synthesis, controls, MEMS fabrication, and optoelectronics. Senior year specializations in VLSI design and in photonics are also available. The department therefore offers a rich environment for development of teaching modules to introduce mechatronic system design into the engineering curriculum. While several courses already in existence focus on design of state-of-the-art MEMS devices, the teaching material we are developing focuses instead on production of standardized devices using well-established technologies available through the MOSIS service and on training designers who will be able to create multidomain "systems on a chip" which will soon become commonplace. Currently the materials we are developing are being used as modules in a variety of courses, including a graduate course on MEMS simulation, an advanced undergraduate course on "silicon programming", and as individual student projects; soon they will be combined into one unified course at the advanced undergraduate-beginning graduate level. The key to making this course feasible is the demonstration of a complete robust design-simulate-fabricate--test cycle which can be implemented in a classroom setting and completed in two quarters. An absolute requirement is that reliable tools with reasonable learning times be employed. We are currently instantiating a prototype for this process, with the help of funding from the U.S. Air Force Wright Labs, with final device testing planned for Spring 2000. Simultaneously, we are developing a suite of VHDL-AMS models which can be used to familiarize students with multidomain behavior and which can also be used as buidling blocks in classroom simulations of complex systems.

2. TOOLS USED; DEVICES STUDIED

Introductory material: we begin with the foundational article by Petersen ([12]). Additional material is based on chapter 5 (micromachining) and chapter 9 (scaling) of [10], while [W15] summarizes current work in MEMS and problems to be solved

Fabrication technology: We are using the fabrication technologies available through the MOSIS CMOS foundry service. In fact, our MEMS structures are fabricated directly from preprocessed (and packaged) CMOS chips using a post processing/packaging wet chemical etch step.[9] Structural features are created on the chip surface using polysilicon, metal, silicon nitride, and SiO2 layers common in a CMOS device process. These structural features are selectively "released" from the silicon substrate using a wet chemical etch process that removes silicon from regions that are photolithographically defined and limited by either the orientation of [111] crystallographic planes or by implantation of p+ etch stop regions. Details of the post processing release step can be found on the MOSIS web page [11]. To facilitate the release process, it is necessary to define two additional CAD layers (called 'open' and 'pstop') in the technology definition files used by common CAD tools like Magic or LEDIT. It should be noted that these logically defined CAD layers are created from multiple mask levels currently available in conventional CMOS fabrication processes. The 'open' layer creates a via that cuts through the overglass, field oxide, and gate oxide layers, leaving a bare silicon surface exposed to ambient conditions. Features drawn in this CAD layer are used to open regions of silicon substrate for etching in the post processing fabrication step. The 'pstop' layer is implemented within the p+ diffusion mask and used to help confine the wet chemical etch process which does not etch highly doped p+ regions in the substrate. As suggested above, these modifications/addition to standard CMOS CAD and fabrication processes can be implemented without change to current CMOS fabrication procedures. Thus, standard CMOS foundry services can be used to fabricate MEMS devices that incorporate both mechanical structures and the electronic circuitry required to control the MEMS devices. Further, the ability to incorporate CMOS logic, analog circuitry and optoelectronic components [4] into the mechatronic chips provides a rich environment for the design and implementation of mixed technology information processing components. Finally, the ability to fabricate these devices using a standardized fabrication process leads to a mixed-domain device technology that builds on the low cost, high reliability, high yield and stable performance metrics that have established conventional VLSI as the technology of choice for employers of electrical and computer engineering students.

Example devices: While the limitation associated with working in a conventional CMOS process may prohibit the design and implementation of some complex mechanical structures, several important mechatronic structures can be implemented with the technology described above. The NIST MEMS library archived on the MOSIS web page includes layouts for a polysilicon heater, a thermal actuator, a micro-hot-plate, and a gas sensor based on the micro-hot-plate. Further, Marshall et al [9] have described an IR emitter based on using the incandescent emission from structures similar to the polysilicon heater. Finally, the technology will easily allow for a variety of electrostatically actuated structures including thin-film diaphragms, cantilever beams, and torsional rotation platforms.

Figure 1 shows the layout of two cantilever beam structures that are designed to be electrostatically actuated deflection mirror structures. These devices form the bases of an optical beam steering system that is capable of directing an optical beam such that it can be reflected off of a large (off chip) fixed mirror and reflected back to an on chip photodetector. The incorporation of photoreceiver circuitry, digital logic, and actuation control circuits will enable the implementation of a robust optical MEMS based information processing system. The development of standard MEMS cell libraries and the design expertise necessary to utilize them in the implementation of mixed technology information processing systems is a major objective of our educational program.

Design tools: As described in [3], we have found that the L-EDIT tools ([14]) work well for design. An alternative, which is available in the public domain, is the LASI system ([2]).

Simulation: simple simulations are carried out with SPICE ([6,7,8]) with translation of electrical parameters into their mechanical equivalents. Students are also introduced to the VHDL (Very High Speed Integrated Circuit Hardware Description Language) extension VHDL-AMS (VHDL with Analog and Mixed Signal Extensions, [5]) to give an integrated simulation environment for components from both the electrical and other domains (Figure 2). A VHDL-AMS simulator is available to educators ([13]). We have also experimented with Mathematica and Ansys ([1,6,7,16]).

3. CONCLUSIONS

We have described a process and supporting tools to support student design, simulation, fabrication, and test of simple mechatronic systems. Use of this material in appropriate courses will provide students with the skills they will need to succeed as complex multidomain systems become commonplace.

REFERENCES

1. *Ansys Primer for Stress Analysis for Revision 4.4,* Swanson Analysis Systems, Inc., 1991.

2.Baker, http://boise.uidaho.edu/langroup/jbaker/wwwbook/winlasi/winlasi.htm.

3. F. Beyette, Jr. and C. Purdy, Tools for integrating multidomain technologies into microelectronic circuit design curricula, *MSE 99*, July 1999, 86-87.

4. Beyette, Stanko, Feld, Mitkas, Wilmsen, Geib, and Choquette, Demonstation and Performance of a CMOS/VCSEL Based Recirculating Sorter", Optical Engineering, 37(1) Jan. 1998, 312-319.

5. Design Automation Standards Committee, IEEE Computer Society, *IEEE Standard VHDL Language Reference Manual (Integrated with VHDL-AMS Changes)*, Std. 1076.1, IEEE, 1997.

6. Gibson and Purdy, Extracting behavioral data from physical descriptions for MEMS for simulation, to appear in *International Journal of Integrated Circuits and Signal Processing.*

7. Gibson, Hare, Beyette, and Purdy, Design automation of MEMS using behavioral modeling, *Ninth Great Lakes Symposium on VLSI*, Ann Arbor, Mich., March 1999.

8. Lo, Berg, Quakkelaar, Simon, Tachiki, Lee, and Pister, Parameterized layout synthesis, extraction and SPICE simulation for MEMS, *ISCAS 96,* May 1996, 481-484.

9. Marshall, Parameswaran, Zaghoul and Gaitan, "High-Level CAD Melds Micromachined Devices with Foundries", IEEE Circuits and Devices, 8(6), Nov.
1992, 10-17.

10. Madou, *Fundamentals of Microfabrication*, CRC Press, 1997.

11. See http://www.mosis.org and follow links for technical support, design
tools, and NIST MEMS Library.

12. Petersen, Silicon as a mechanical material, *IEEE Proceedings* 70 (5), May 1982, 420-457.

13. SEAMS Simulator Project, Distributed Processing Laboratory, ECECS Department, University of Cincinnati, 1998.

14. Tanner Research, Inc., *Tanner Tools User's Manual*, 1996.

15. Wise, VLSI circuit challenges in microelectromechanical systems, *Proc. Ninth Great Lakes Symposium on VLSI*, March 1999.

16. Wolfram, *Mathematica: A System for Doing Mathematics by Computer*, Addison-Wesley, 1991.

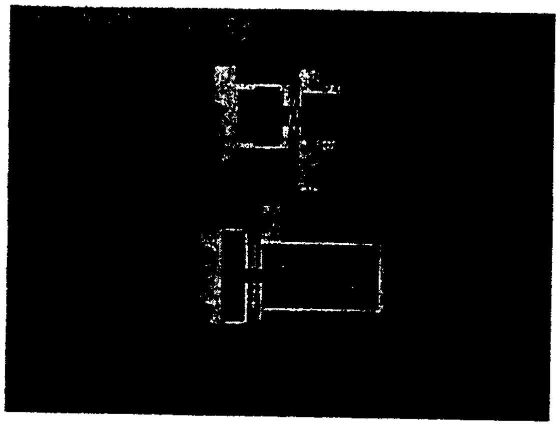


Figure 1. Example cantilever beam system.

```
entity mechsensor is
  port(f: in force; c: out capacitance);
end mechsensor;

architecture twopart of mechsensor is
  component mechinput . . . ;
  component capacitor . . .;  signal y: distance;
  begin
  m1: mechinput port map (fb=>f, x=>y);
  c1: capacitor portmap (xb=>y,cb=>c);
end twopart;
```

Figure 2. Portions of VHDL-AMS mechanical sensor model.

The Strengths and Weaknesses of VHDL-AMS as a Tool for MEMS Development

Dennis Gibson and Carla Purdy

September 2000

Abstract

Currently, inadequate languages and tools for modeling and simulation of MEMS[2] components and systems force designers to resort to physical prototyping. This paper discusses the basic requirements for MEMS modeling and simulation, and reviews currently available simulators with respect to these requirements. In particular, this paper discusses the strengths that VHDL-AMS has to offer to the MEMS designer for component and system modeling. In addition, it provides an understanding of the areas in which VHDL-AMS falls short and needs to be improved in order to be a valuable tool.

Introduction

Today's state-of-the-art digital design systems typically support many different levels of design activity. At the highest levels in the design process, structural system descriptions and high-level language component descriptions promote system-level thinking, while at lower levels sophisticated simulation tools provide detailed descriptions of physical behavior. Fabrication processes are well-characterized, and parameters from these processes are routinely collected for use in low-level simulations. These differing levels of design and simulation activity are tied together by sophisticated analysis, synthesis and verification tools. Thus digital systems can be rapidly prototyped and modified, and large-scale commercialization of these systems has become a reality. In contrast, the current state of the art for production of multi-energy domain systems is much more primitive. MEMS fabrication processes are extremely varied and, in consequence, much less standardized than electrical processes [7]. Robust simulation tools which can handle interacting energy domains efficiently, such as MEMCAD, IntelliCAD, and the CFDRC tools, are just beginning to become available [10], as are some tools for use in university research [9,12]. MEMS component libraries do exist, but they tend to consist of individual components which must be laboriously integrated into working systems [6,11]. And techniques for macro-level MEMS design are almost nonexistent.

Ideal MEMS simulation language

It has been stated that the three main computational challenges for efficient MEMS simulation are:

[2] Here we will use the term MEMS to mean any multi-energy domain micro-system, including but not limited to electromechanical systems.

1) developing faster algorithms for computing surface forces due to fields or fluids exterior to geometrically complex, flexible three-dimensional structures,
2) handling complicated interaction between energy domains, and
3) developing dynamic models that permit rapid simulation of system performance under a wide variety of inputs and scenarios [18].

Challenge one is outside óf any language, but is an issue to be dealt with by any MEMS simulator. Developing faster algorithms is a challenge for MEMS simulation, but it is not necessarily an issue for a design language. Thus, our emphasis will be on challenges 2 and 3. In addition, the ideal language needs to address key hardware/software (HW/SW) issues such as modularity, reusability, maintainability, abstraction and IP protection.

Current approaches to MEMS design and simulation
Here we will discuss three possible approaches to MEMS design and simulation.

- One first approach to MEMS simulation is to create a simulator based on coupling, through a common interface, a number of simulation tools, each optimized for a particular energy domain (such as SPICE for electrical, ANSYS for mechanical, etc.). MEMCAD [19] is an example of such an approach. This approach may handle challenge 2 with a divide and conquer strategy. Each component in model is divided into separate energy domains. Each energy domain is calculated in parallel by each of the specific energy domain solvers at each step in the simulation. The key to the third challenge in this approach is to use a reduced order model [17] (or macromodel [18]). A reduced order model is an abstract notation for a simplified set of equations to represent the terminal characteristics of a subsystem. It is used to represent static and dynamic behavior to an acceptable level of fidelity, emphasizing terminal characteristics. Accuracy is measured with respect to the original subsystem behavior [17]. MEMCAD 3D models may be used to generate the reduced order models that are be represented as input files to some ODE solver.

- A second approach involves translating each set of energy domain equations into an equivalent electrical circuit. For instance, a system which has components in the electrical and mechanical energy domains could translate the mechanical components into the electrical domain. The model can then be simulated with any electrical energy domain solver (i.e., SPICE). Once the translation is accomplished, then challenges two and three are easily accomplished. However, the primary disadvantages to this approach are [21]:
  - The introduction of spurious energy sources, when using controlled sources to model nonlinear components.
  - Usually, all components are linearized around an operating (bias) point limiting the validity of these models to small-signal analysis.
  - The approach is constrained by the choice of circuit elements found in the SPICE library.
  - The (often) difficult task of finding an equivalent circuit.

- A third approach is to create a single solver that incorporates multiple energy solution domains. SUGAR [9] is an example of such an approach. When a system is composed of hundreds or thousands of components, traditional FEA based tools are not feasible for simulation. Nodal analysis has been used in circuit analysis and is accomplished by dividing a circuit into a number of individual devices. Each device has its own model in the form of ODEs Devices are linked at their terminals or nodes and can be solved as a system of nonlinear ODEs using nodal analysis [15]. Nodal analysis allows for the coupling of energy domains and for simplified models to be input. Thus, challenges two and three are able to be met to some degree.

All of the approaches, however, concentrate mainly on challenge one above. What we wish to explore here is how to define a language that allows us to meet challenges two and three, relatively independent of how the underlying simulation will function. In particular, we will look at how VHDL-AMS can provide the necessary interface.

The VHDL-AMS language

Recently, however, researchers have begun to succeed in extending digital design methodologies to the analog domain, with "mixed signal" design and simulation tools being developed. One such tool is the VHDL-AMS (Very High Speed Hardware Description Language with Analog and Mixed Signal extensions) [1]. Here we show how this language can in fact be used to support modular design, component reuse, and hierarchical development, not only of mixed signal systems but also of multi-domain, or MEMS, systems.

VHDL-AMS is a structured programming language with origins in the Ada language. VHDL-AMS is an extension to the analog domain of VHDL, which was originally designed for simulation of digital circuits and systems and which has also been used extensively in digital circuit synthesis. As in VHDL, a description consists of an "entity" specifying inputs and outputs, as well as associated "architectures" describing, possibly in different styles and at different levels of abstraction, how the inputs will be transformed into the outputs. VHDL-AMS is, in fact, not limited to the electrical domain. It has the ability to handle any algebraic or ordinary differential equation or any system of such equations. VHDL-AMS does not yet incorporate either finite element methods or distributed parameter calculations. However, for a given problem, it is possible to define a set of elements and their associated equations offline and, using piecewise linear approximations, to deal not only with lumped parameters but also with distributed parameters. In addition, because it must deal with both digital and analog "events", VHDL-AMS can handle both discrete and continuous phenomena. Currently, complex VHDL-AMS SPICE_type models for transistors and other electrical components are becoming available [5]. But VHDL-AMS can also be used at much higher levels of abstraction to specify connections between system components. A VHDL-AMS "compiler" can output intermediate code for any simulator or simulators, as long as certain rules are followed. For example, the compilation step of the SEAMS simulator used for this work [12] has options to output C++, VHDL or code for parallel execution.

VHDL-AMS solutions to challenges

VHDL-AMS is a good starting point. For instance, systems such as SUGAR [9] are starting from ground zero by building their system on top of MATLAB. VHDL-AMS has been dealing with the electrical energy domain for quite some time. This electrical energy domain is well understood. In addition, VHDL-AMS has been developed to handle very complex systems (the modern microprocessor is indeed a complex system). It is now necessary to examine how VHDL-AMS is able to adapt itself to the challenges posed for an ideal MEMS simulation language.

Solution for challenge 1. The first challenge was previously stated as a need for more efficient algorithms. This challenge is more suited for a simulator for the language. However, if the language simulator is built upon a proprietary product such as MATLAB, then incorporating these new and more efficient algorithms becomes a problem. Thus, it appears to be advantageous at this point to avoid using systems built upon proprietary products.

Currently, there are few simulators available and many are still in development. SABER [22], MEMCAD, SUGAR and SEAMS [12] are examples of available simulators.

Solution for challenge 2. The second challenge was a means to handle the complicated interactions between the different energy domains. One way that VHDL-AMS may approach this problem is through nodal analysis, similar to the approach that SUGAR uses. Since VHDL-AMS has the ability to solve systems of ODEs, and with the use of energy domain packages in VHDL-AMS, a means to represent nodes or terminals is available. The VHDL-AMS standard supports the aggregation of common definitions within a language construct called a package. Models may reference these common packages to make use of these common definitions.

The VHDL-AMS "nature" construct defines a template for terminal definitions. Natures define the "across" and "through" aspects of conserved energy connections. Terminals provide connection points where the conservation of energy laws are enforced. These laws state that the sum of all across quantities around a closed path must equal to zero, and that the sum of all through quantities at a node must equal zero when the system is in equilibrium [17]. Common packages have been developed to support the modeling of any combination of the following systems: electrical, mechanical, fluidic, thermal and radiant [17].

Solution for challenge 3. Creating reduced-order models or macromodels by hand is a tedious and lengthy process. Research is currently being done to automate this procedure. One product of such research is AútoMM [20], which automatically generates a macromodel equation of motion. This equation can then be expressed in the form of a netlist or input file for any number of system level simulators or ODE solvers.

It is clear that VHDL-AMS may be used as one of the forms. VHDL-AMS allows designers to model any system that can be represented by ODEs at any level of abstraction and supports both conservative and nonconservative (signal flow) systems [17]. Therefore, VHDL-AMS has the capability to represent these reduced order models.

Solution to other challenges. Modularity may be accomplished by designing a system using component instantiation. In other words, a system may be comprised of individual devices which have reduced order models, and are assembled by detailing the terminal connections. VHDL-AMS models have the ability to be reusable. However, for this to be realized, they must be parameterized and have their design intent documented [17]. Multiple levels of abstraction are available in VHDL-AMS since specification may be described by [14]:

1) a process (algorithmic style)
2) an equation (functional style)
3) instantiation of coupled components (structural style)

VHDL-AMS strengths as a MEMS simulation language

VHDL-AMS has many features which enable it to be a suitable design language for MEMS.

- It has the ability to handle discontinuity in models. This capability gives the modeler the ability to details into the model that were not possible before. For example, chaos theory may be exploited in the mechanical domain. Modeling fractures in mechanical elements is now possible. Hysterisis of electrically actuated beams may be modeled, where the pull-in voltage of beam and its release voltage are not the same [3].

- There is a smooth transition between models at low levels of abstraction to high levels of abstraction. For instance, a highly detailed low level component model is easily replaced with a high level macromodel.

- The ability to model both conservative systems as well as nonconservative systems is a strength. VHDL-AMS has the ability to define quantities to obey/disobey conservation laws and even to mix the two types of quantities. For example, one can mix signal flow (nonconservative) and circuit (conservative) descriptions together. The ability to model both types of systems gives the ability to solve many linear programming problems.

- There exists a foreign language interface to VHDL-AMS. This interface allows for functions, procedures or entities to be coded in a language other than VHDL-AMS but be added to a system using the "FOREIGN" attribute.

VHDL-AMS weaknesses as a MEMS simulation language

There are few weaknesses to VHDL-AMS which need to be stated. The following is a list of such weaknesses:

- The largest drawback to VHDL-AMS is the lack of simulators which support it. Even the simulators which do support it, usually do not support all the features of VHDL-AMS.

- Another drawback to VHDL-AMS is the inability to do symbolic computation. It is only able to do number "crunching" type of tasks.

- VHDL-AMS is currently limited to only expressing ODEs and algebraic equations. It cannot handle partial differential equations. It can only deal with differentiation and integration relative to time. MEMS requires differentiation and integration over physical entities such as distance. In addition, it only deals with first order differentiation and integration. It cannot deal with differentiation and integration over multiple variables.

- VHDL-AMS is limited in the frequency domain. It only possesses Laplace and Z domain transfer functions and noise sources. Furthermore, all frequency modeling is translated to the time domain functions for the simulator. There is no provision for obtaining output of simulation in terms of the frequency domain.

- There is no built in engineering unit conversion capability. The modeler must explicitly create unit conversions.

Conclusion

By incorporating VHDL-AMS into emerging MEMS design systems, it should eventually be possible to develop design methodologies similar to current digital design methodologies and hence to more easily achieve commercialization for these much more complex systems. Thus the many benefits which MEMS systems promise [8] will become realizable. Our remarks also apply, of course, to similar HDL's such as Verilog and its extensions.

More development of the VHDL-AMS language and extension of current VHDL-AMS simulators to include all VHDL-AMS features would be both a sound leveraging of current capabilities and a sound investment which will lead to usable MEMS design systems and eventually to rapid prototyping and economic production of multi-energy domain systems.

References

1. Design Automation Standards Committee, IEEE Computer Society, *IEEE Standard VHDL Language Reference Manual (Integrated with VHDL-AMS Changes)*, Std. 1076.1, IEEE, 1997.

2. S. Ghosh and N. Giambiasi, Language barriers in hardware design?, *Circuits and Devices*, Sept. 1999, 25-40.

3. D. Gibson, A. Hare, F. Beyette, Jr., and C. Purdy, Design automation of MEMS systems using behavioral modeling, *Ninth Great Lakes Symposium on VLSI,* Ann Arbor, Mich. (ed. R.J. Lomax and P. Mazumder), March 1999, pp. 266-269.

4. D. Gibson and C. Purdy, Extracting behavioral data from physical descriptions of MEMS for simulation, *Analog Integrated Circuits and Signal Processing* 20, 1999, 227-238.

5.V.R. KasulaSrinivas, Modeling semiconductor devices using the VHDL-AMS language, M.S. thesis, University of Cincinnati, 1999.

6. D.A. Koester, R. Mahadevan, A. Shishkoff, and K.W. Markus, *Smart-MUMPS Design Handbook Including MUMPS Intorduction and Design Rules (rev. 4)*, MEMS Technology Applications Center, MCNC, 1996.

7. K.W. Markus and K.J. Gabriel, MEMS: the systems function revolution, *IEEE Computer* 32 (10), 1999, 25-31.

8. K.E. Petersen, Silicon as a mechanical material, *IEEE Proceedings* 70 (5), May 1982, 420-457.

9. K. Pister, SUGAR V1.0, http://www-bsac.EECS.Berkeley.edu/~cfm/

10. S.D. Senturia, Simulation and design of microsystems: a 10-year perspective, *Sensors and Actuators* A 67, 1998, 1-7.

11. *Tanner Tools User's Manual*, Tanner Research, 1996.

12. University of Cincinnati, ECECS Department, Distributed Processing Laboratory, SEAMS simulator project, http://www.ececs.uc.edu/~hcarter

13. *YSI Precision Thermistors & Probes*, YSI, Incorporated, 1997.

14. KANDIS - A Tool for System-Level Specification and Design of Mixed-Signal Systems Christoph Grimm, Peter Oehler, Isidoros Kanakis, Klaus Waldschmidt BMAS '97, Washington DC, USA, 20. - 21. October 1997 (c) VIUF, 1997

15. K.S. Pister, Ningning Zhou, and Jason Vaughn Clark, Sugar: Nodal Analysis for MEMS Design using Sugar v 0.5.

16. Matlab, Matlab High-Performance Numeric Computation and Visualization Software Reference Guide, The Math Works, Inc., 24 Prime Park Way Natick, Mass, 1992.

17. James P. Hanna and Robert G. Hillman, A Common Basis for Mixed-Technology Micro-System Modeling, Technical Proceedings 1999 International Conference on Modeling and Simulation of Microsystems - Technical Proceedings 1999 International Conference on Modeling and Simulation of Microsystems MSM 99San Juan, Puerto Rico, USA April 19-21, 1999

18. Stephen D. Senturia, Narayan Aluru and Jacob White, Simulating the Behavior of MEMS Devices: Computational Methods and Needs, IEEE Computational Science and Engineering, Jan-Mar 1997, pp. 30-43.

19. Microcosm website, http://www.memcad.com/, 2000.

20. N. R. Swart, S. F. Bart, M. H. Zaman, M. Mariappan, J. R. Gilbert, and D. Murphy, AutoMM: Automatic Generation of Dynamic Macromodels for MEMS devices, Analog Devices, Inc., Cambridge, MA 02139, Microcosm Technologies, Inc., Cambridge, MA 02142

21. http://www.vhdl-ams.com/Microcosm/VHDL1076.1/ default.html, MEMS Modeling Examples, Bart F. Romanowicz, 2000.

22. http://www.analogy.com/products/Simulation/simulation.htm#Saber

Characterization of a monolithic silicon MEMS technology in standard CMOS process

Kuntao Ye and Fred R. Beyette Jr.

Department of Electrical and Computer Engineering & Computer Science,
University of Cincinnati,
Cincinnati, OH 45221-0031, USA

ABSTRACT

This paper presents a comparative analysis between two specific post-processing techniques (RIE dry etching and TMAH wet etching) that are suitable for implementing a monolithic CMOS compatible MEMS fabrication technology. Further, an experimental investigation is presented which details the fabrication of MEMS structures by TMAH post etching of a CMOS chip fabricated in a standard AMI 1.5μm CMOS process. Finally this paper provides future designers with experimental data that will allow for the design and fabrication of simple MEMS structures using a standard CMOS process.

1. INTRODUCTION

Research in the area of micro-electromechanical systems (MEMS) technology has developed significantly since Kurt Petersen first presented MEMS fabrication techniques on silicon wafers near twenty years ago[1]. While MEMS technologies have had a major impact on the design of electronic systems, the complete integration of the MEMS technology and IC technology has yet to be realized. Many researchers have been working to modify the IC fabrication process to implement a CMOS compatible MEMS technology. For universities, government laboratories and small businesses that do not have an in-house custom integrated circuit (IC) fabrication facility, the ability implement MEMS structure with a standard commercial CMOS fabrication process is extremely important. Fortunately, research conducted over the last decade has produced several post processing techniques that allow the fabrication of MEMS devices from chips fabricated in a standard CMOS IC process[2,3].

At the same time the development of CAD tool that are capable of simulation, synthesis and extraction in mixed-domain environment (including MEMS devices) has become a critical need for system designers. Recently, Gibson et al. have compared the use of different simulation tools for simple cantilever beams in fabricated using the MUMPS[4] process. Additionally, Mukherjee et al. have analyzed the comb drive structure and present a technique for doing mixed-domain circuit simulation[5].

This paper presents a comparative analysis between two general types of CMOS post processing that are suitable for MEMS fabrication. Additionally, an experimental investigation of MEMS structures fabricated by TMAH etch post processing is presented. The intent of this work is to provide designers with an the experimental based design methodology that utilizes a standard CMOS process. Ultimately experimental data from this study may facilitate the development of CAD tools capable of mixed-domain simulation, synthesis and extraction.

2. COMPARISON BETWEEN TWO SPECIFIC POST PROCESSING

There are many techniques that can be used to fabricate CMOS compatible MEMS devices. Several of the most common methods involve post processing standard CMOS chips. We can categorize post processing techniques into two general classes: dry etching and wet etching. As described in the following subsections, each category yields different results.

2.1Dry etching post processing

Fig.1 and Fig.2 show device cross sections for two different dry etch post processing techniques developed by researchers at Carnegie Mellon University[6,7]. Fig.1. shows a three-step, maskless RIE dry etching procedure. Fig.1.a is a schematic cross section view of the chip before dry etching. First a two-step

CHF_3/O_2 etch is used to cuts through the thick oxide layers. Fig.1.b shows the first step where the over glass layer has been etched away under a relatively high pressure of 50 mtorr and high power of 100w. Fig.1.c shows the second step where selective CHF_3/O_2 RIE etching of the field oxide is done using metal-3 aluminum as an etch mask. Relatively lower pressure (25 mtorr) and lower power (50w) are used to avoid oxide 'stringers' at the base of the sidewall. Once the oxide has been removed to the substrate, a dry isotropic RIE step using SF_6/O_2 is performed to undercut (and release) the microstructure.
To maximize designability, D. F. Guillou et al. investigated an alternative RIE dry post processing[7]. Fig.2. illustrates this process. The first two steps still use CHF_3/O_2 under nominal pressure and power to etch away the over glass layer and anisotropically etch down to the middle of the metal-1 layer. In the third step, the flow rate of CHF_3 is reduced allowing for controlled etching through the polysilicon layer. Note that after this third step, the bottom of the sidewall has not reach the substrate. In the final step, dry etching with SF_6/O_2 is done to etch away the poly-silicon layer and thus release the MEMS structure.

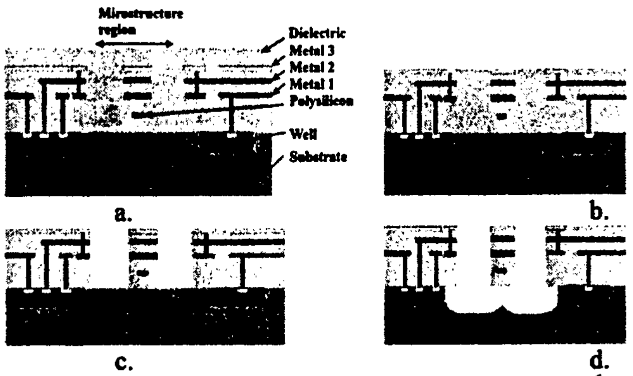


Fig.1. Dry etching post process for substrate removal.[6]

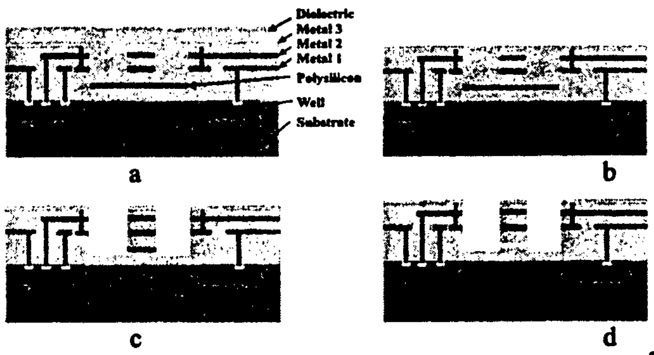


Fig.2. Dry etching post process for poly silicon layer removal.[7]

Although dry etch post processing does not suffer from the "sticktion' problems that is a common difficulty encountered with wet etching techniques, the RIE equipment required is significantly more complicated and expensive to maintain. Further, the dry etching techniques require multiple etch steps that must be tightly controlled to insure successful release of the MEMS structure with out collateral damage to the CMOS circuitry that is monolithically integrated onto the same chip.

2.2Wet etching post processing

Wet etch post processing techniques have also been extensively studied through the last decades as an inexpensive release process for microstructures on a standard CMOS chips. The wet etching post process utilizes a much simpler experimental setup that is cheaper and easier to perform. Fig3. shows device cross sections that illustrate the steps of a typical wet etch post process. Fig.3.a is a schematic view of a standard CMOS chip without direct opens. By modifying the CMOS technology file used by the VLSI CAD /ayout tool, it is possible to create virtual fabrication layers that combine all of the via and overglass opening features in a fabrication process. Using such a virtual "opening" layer it is possible to realize direct openings to the substrate on a standard CMOS chip[2]. Fig.3.b shows the cross sectional view of a CMOS chip with these direct openings that are created as a part of the conventional CMOS fabrication process.

After CMOS fabrication is completed, the chip is submerged in a wet silicon etch solution for a period of time that is sufficient to completely release the MEMS structure (Fig.3.c). Compared to the dry etching approach, the wet etch post processing technique is relatively simple with only one etch step that can be implemented using equipment that is readily available in any microfabrication laboratory.

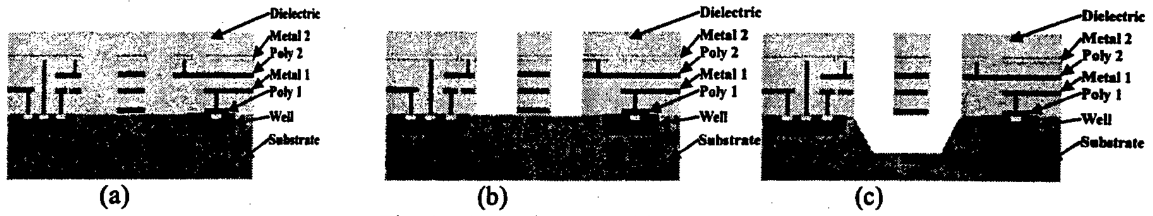


Fig. 3 schematic wet etching process

In a wet etch post processing technique, choosing the right etch solution is extremely important. The two most common etchant solutions are ethylenediamine-pyrocatechol (EDP) and silicon doped tetramethyl ammonium hydroxide (TMAH). Both of these etch solution are preferred over KOH which suffers from the problem of K^+ contamination that degrades the performance of CMOS circuits monolithically integrated with the MEMS structures. Because they etch silicon without attacking CMOS dielectric layers and/or metal layers, both of EDP and silicon doped TMAH solution are compatible with IC technology.

Although EDP is a common etchant used in industry, it is a harmful carcinogenic chemical and thus, many semiconductor processing facilities have elected not to allow the use of EDP. EDP is NOT typically recommended for work in universities where students are learning to use chemical etchants[8].

As suggested above, TMAH must be doped with silicon in order to selectively etch silicon while keeping exposed metal lines and pads intact during the etching process.[9-12] In an etchstudy, we found that 375ml 5wt% TMAH water solution, doped with 5.8 gram of silicon powder and 1.8 gram of ammonium peroxodisulfate yields a silicon etch rate of roughly 0.9 um/min at 85 °C. Further, the solution will not attack aluminum layers. We use this etchant recipe through out our wet etching post process. Note: the etch solution is surrounded by a constant temperature water bath and stirred with a magnetic stirrer to control the temperature during the etching process.

Another important issue to consider with wet etching processes is the drying technique used after the chip is removed from the etchant. The choosing the proper drying technique, it is possible to significantly minimized damage to fragile MEMS structures. Chang-Jin Kim et al. have compared 5 different of dry techniques[13] including:

- Evaporation drying with deionized (DI) water.
- Evaporation drying with methanol.
- Sublimation drying with t-butyl alcohol.
- Sublimation drying with p-dichloreobenzene
- Super critical drying with CO_2.

We have chosen evaporation drying with methanol because it uses a very simple apparatus and methanol has much smaller surface tension than water. Thus the "sticktion" problem has a much smaller chance of damaging fragile structures.

3. THE DESIGN AND SIMULATION

From the above discussion, we clearly see that wet etch post processing can be performed on CMOS chips that are fabricated using a standard CMOS process. Further, wet etching techniques have several desirable characteristics including ease of implementation relatively low cost of both equipement and etchant chemicals. Ideally wet etch post processing of CMOS chips could become a popular technology that enables the design and fabrication of MEMS based microstructures. In this section, we present a detailed description of our wet etch post processing technique. The wet chemical etch process has been developed to enable fabrication of MEMS devices on standard CMOS chips.

To illustrate our approach a CMOS chip has been fabricated using the AMI 1.5 μm CMOS process available through the MOSIS service. The MOSIS AMI 1.5u process layers are shown in Fig3. We used the Magic technology file SCNA_MEMS.80.tech27 from MOSIS to layout 41 different microstructures. All of the test structures are square shape suspended plates with support necks surrounded by direct openings to the silicon surface. Fig 4 shows the three types of structures included on the chip. We have varied the structure neck length (a), neck width (b), and plate width (c).

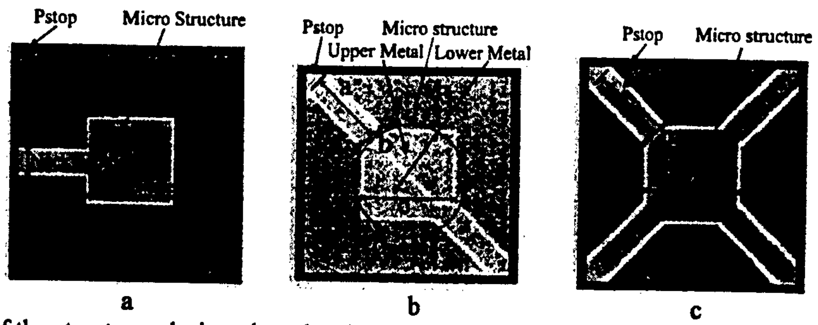


Fig.4 Types of the structures designed on the chip (a: capacitive type 1, b: capacitive type 2, c:capacitive type 3)

Below is a brief description of each device type:

Capacitive Type 1 (CT1): As shown in Fig 4a, this microstructure consists of a square plate and a single neck composed of a metal-1 layer sandwiched between oxide layers.

Capacitive Type 2 (CT2): As shown in Fig 4b this microstructure consists of a square plate and two necks composed of a metal-1 layer sandwiched between oxide layers. As shown, the two necks are oriented 45 degree to the plate edge.

Capacitive Type 3 (CT3): As shown in Fig 4c this microstructure consists of a square plate and four necks composed of a metal 1 layer sandwiched between oxide layers. As with the type 2 structure, the necks are oriented 45 degree to the plate edge.

As shown in tables 1-3, the test chip includes a wide variety of device sizes for each structure type.

Table 1: CT1 parameters (um)

c b a	60	100	200	300
40	11,20	11,20	20	20
50	11,20	11		
60		11,20		

Table 2: CT2 parameters(um)

c b a	60	100	200	300
40	11,20	11,20	11,20	20
50	11,20	11		
60		11,20		

Table 3: CT3 parameters(um)

C B a	60	100	200	300
40	11,20	11,20	11,20	20
50	11,20	11		
60		11,20		

4. EXPERIMENTAL POST PROCESSING PROCEDURE AND RESULTS

After the chip is returned from the CMOS foundry, we must perform the post process wet etching step to release the MEMS structures. In order to develop a wet etching process that is compatible with a CMOS chip fabricated through the MOSIS foundry service, we have first performed the wet etching on an oxidized (100) p-type test wafer. To provide a test wafer that matches the actual device the oxide layer of the test wafer is patterned to incorporate openings with the exact sizes and shapes as the openings on the MOSIS chip.

Before beginning the post process etching it is necessary to prepare the silicon doped TMAH solution. Starting with 375 ml of 5wt% TMAH, the solution temperature is brought up to 85 °C with the constant temperature bath. Next 5.8 gram of silicon powder is slowly added to the TMAH as a magnetic stir bar gently mixes the solution. Finally, 1.8 gram of ammonium perfulfate is added to the solution. Note: a white precipitate forms as the ammonium perfulfate is mixed into the solution. After the about 30 minutes, the precipitate dissolves indicating that the solution is ready for etching.

Before performing the wet etch it in necessary to remove the native oxide formed on the exposed silicon surface. This is accomplished by dipping the CMOS chip in 20:1 HF for 1 minute. After native oxide removal, the chip is transferred into the etch solution. A magnetic stirrer provides gentle agitation of the solution which is held at a constant temperature of 85 °C. After the etch step has completely released the MEMS structures, care is taken to minimize "sticktion" that can occur during sample drying. First to terminate the etch process, the chip is removed from etch solution and immerse in DI water for 5 minutes.

Then to reduce the effects of surface tension between the water and released structure, the chip is transferred from the DI water to a 5 minute methanol bath. Finally, the sample is dried for 10 minutes in a 60 °C drying oven.

To characterize the etch process, the test wafer was dipped into the etch solution and removed at regular intervals in order to measure the etch depth and undercut rate. Fig.9 shows the etch depth which was measured optically using the difference in focus depth between the top and bottom surfaces. Undercut rates were determined from the images shown in fig. 10. As shown in the figure, noticeable undercutting is observed after 30 minutes of etching. We measured the undercut length from the corner of the beam to the nearest silicon region along the directions shown in Fig.10. The result of this undercut measurement are plotted versus etch time in Fig.11. Based on the graph in figure 11, a 5 hour etch time will guarantee that all of the structures on the test chip are released.

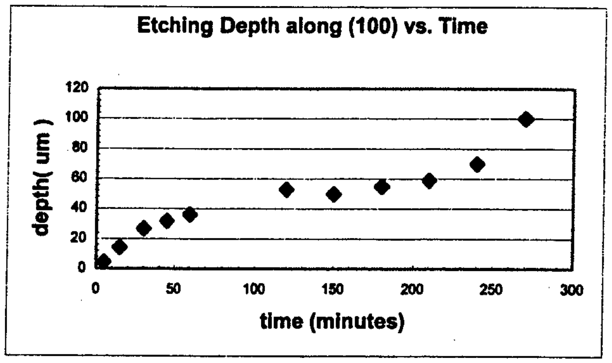
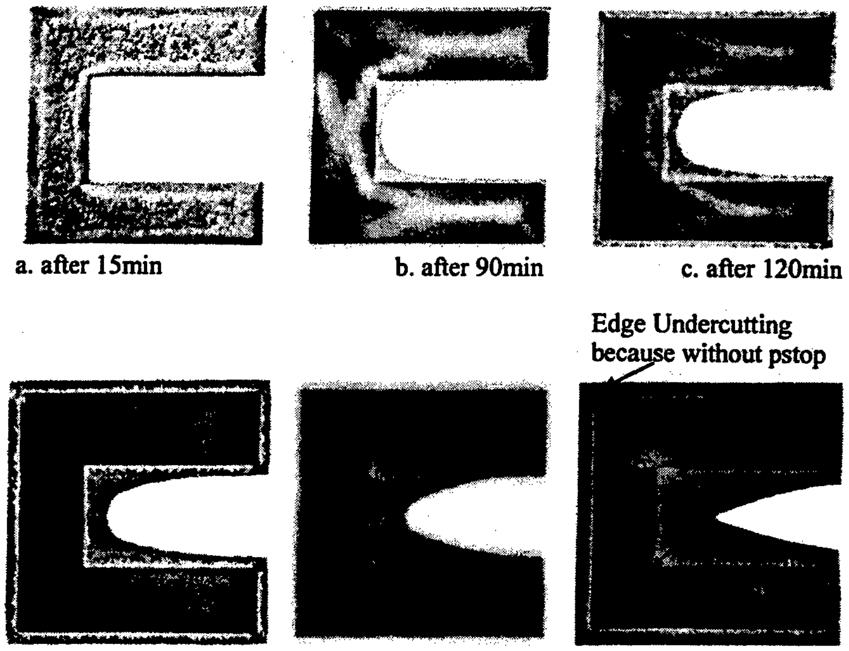


Fig 9 etching depth vs. time



a. after 15min b. after 90min c. after 120min

d. after 180min e. after 240min f. after 270min

Fig.10 Undercutting evolving procedure (actual structure area size is 160um x 160 um)

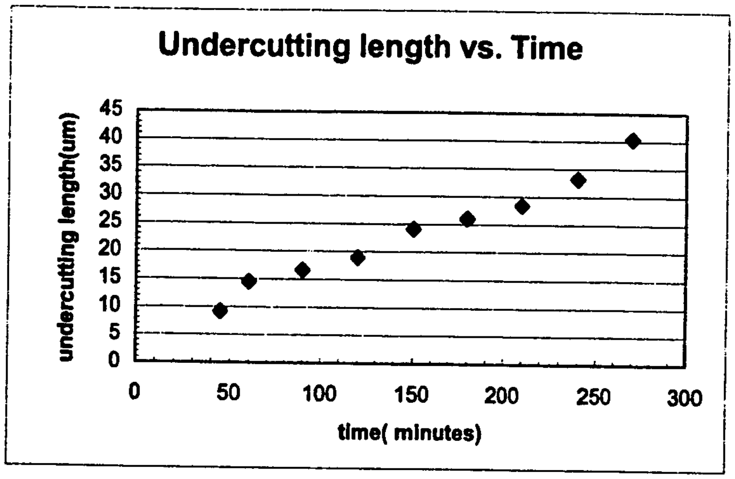
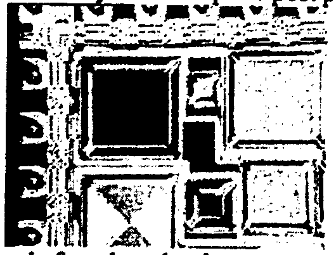


Fig 11. Undercut measurement result

To verify that the silicon doped TMAH etch does not attack aluminum, we etched a small piece of aluminum-coated silicon wafer in parallel with the release of structures on a test wafer. The aluminum layer thickness was measured before and after the etch step using a Dektek II profilometer. No noticeable reduction in aluminum film thickness was detected after the 5 hours etch process. Thus, we are confident that Silicon doped TMAH is a suitable wet chemical etch for releasing MEMS structures implemented in a standard CMOS MEMS process.

After the post processing procedure was completed, the chip was examined under a microscope. Fig. 12 shows images taken before and after the wet etch post processing steps were completed. As expected, the 5 hour etch time was sufficient to release all of the structures on the chip. Unfortunately, the chip shows two potential problems. First, the after post processing images show that the bonding wires have been broken off. This is probably due to overly aggressive stirring during the etch process. Second, the color variation across the platform feature of the larger devices suggests that the platforms may be curling as a result of strain inside of the laminated oxide and metal 1 layer structure. To alleviate this problem, it may be necessary to develop the post processing steps required to deposit a strain compensation layer.



a. before the releasing b after the releasing

c. before the releasing d. after the releasing

Fig. 12 Photomicrograph showing the CMOS chip before and after the post processing steps. (The actual size of the photo area is L x H = 1230 μm x 949 μm. The size of the OT1 area is 285 μm x 218 μm. The actual size of the OT2 area is 403 x μm x 365 μm.5. Disccussion And Conclusion

The results described above, show that we have successfully characterized a monolithic, silicon MEMS technology that is compatible with a standard CMOS process. We have measured both the etch rate, and the undercut rate for silicon wafers that include CMOS circuitry. From the result photos of Fig.13, it can be seen that the surface of the metal 1 layer is rough and thus would not be a good choice for a deflectable optical mirror. While problems like these highlight the short comings that are frequently a part of new device technologies, it is clear that the process described here has considerable

a b c

Fig.13. The successfully release structure of CP1, CP2, and CP3. (actual area of the photo area is L x H = 285um x 218 um; CT1 with a = 50um, b=20um, c=60um; CT2 with a= 40um b=20um c=60um), CT3 with a= 40 um, b=20um, c=100um)

REFEFRENCES

1. Kurt E. Petersen, "*Silicon as a Mechanical Material*", Proceedings of the IEEE, vol. 70, No. 5, May 1982, pp. 420-557

2. Janet C. Marshall, M. Parameswaran, Mona E. Zaghloul, and Michael Ganan, "*High-Level CAD Melds Micromachined Devices with Foundries*", Circuits and Devices, November, 1992, pp10-17

3. Xu Zhu, David W. Greve and Gary K. Fedder, "*Characterization of Silicon Isotropic Etch by Inductively Coupled Plasma Etch in Post-CMOS Processing*", The 13th Annual International Micro Electro Mechanical Systems Conference Miyazaki, Japan, Jan 23-27, 2000, pp. 568-573.

4. Dennis Gibson and Carla Purdy, "*Extracting Behavioral Data from Physical Descriptions of MEMS for Simulation*", Journal of VLSI Signal Processing 22, pp. 135-146(1999)

5. Tamal Mukherjee, Gary K. Fedder, "*Hierarchical Mixed-Domain Circuit Simulation, Synthesis and Extraction Methodology for MEMS*", Journal of VLSI Signal Processing 21, 233-249(1999)

6. G.K.Fedder, S. Santhanam, M.L. Reed, S.C. Eagle, D. F. Guillou, M.S.-C. Lu, "*Laminated high-aspect-ratio microstructures in a conventional CMOS process*", Sensors and Actuators A 57(1996) pp. 103-110

7. D.F. Guillou, S. Santhanam, L.R. Carley, "*Laminated, sacrificial-poly MEMS technology in standard CMOS*", Sensors and Actruators 85 (2000) pp. 346-355

8. http://www.mosis.com/Technical/Designsupport/nist-mems-1.html

9. Osamu Tabata, "*pH-contralled TMAH etchants for silicon micromaching*", Sensors and Actuators A 53 (1996) pp.335-339

10. Kazuo Sato, Mitshuhiro Shikida, Takashi Yamashiro, Kazuo Asaumi, Yasuroh Iriye, Masaharu Yamamoto, "*Anisotropic etching rates of single-crystal silicon for TMAH water solution as a function of crystallographic orientation*", Sensors and Actuators 73(1999) pp.131-137

11. Mitsuhiro Shikida, Kazuo Sato, Kenji Tokoro, Daisuke Uchikawa, "*Differences in anisotropic etching properties of KOH and TMAH solutions*",

12. Yan, Guizhen, Chan, Philip C.H., Hsing, I-Ming, Sharma, Rajnish K., Sin, Johnny K.O., Wang Yangyuan, "*An improved TMAH Si-etching solution without attacking exposed aluminum*", Sensors and Actuators A: Physical Vol. 89, Issue 1-2, (2001) pp. 135-141

13. Chang-Jin Kim, John Y. Kim, Balaji Sridharan, "*Comparative evaluation of drying techniques for surface micromachining*", Sensors and Actuators A 64 (1998) pp17-26

TASK 21

Optimized Global Information Compression Methodologies and Implementation of Enhanced C41SR System Integration

Principal Investigator:	Prof. Frank Scarpino
Institution:	University of Dayton, Dayton, Ohio
Period of Performance:	Octgober 18, 2000 to June 10, 2002
Task:	Create and demonstrate a FPGA-based hardware accelerator for audio-visual compression with potential use in the Global Information environment.

21

Image Compression Using Super Efficient Harr Transforms

University of Dayton Research Institute

Dr. Frank Scarpino
Joseph Fieler
Shawn Nichols
Jeff Shafer
William Turri

July, 2002

Abstract

In today's ever-changing multimedia world the need for fast, effective compression schemes is absolutely necessary. The success of these schemes is based on their ability to compress large amounts of image data. Most modern applications, such as video teleconferencing, medical imaging, and military applications, contain high-resolution image frames and possess sizeable quantities of data. Subsequently, these images require mass amounts of memory for storage, and high-speed networks for transmission. Herein lies the basis for this study, which illustrates how wavelets are utilized to eradicate these problems in a digital hardware environment. For the purposes of this investigation VDHL is used in conjunction with a SLAAC1-V board.

General Wavelet Transformation Theory

Wavelet transforms prepare an image for compression by creating a sparse, multi-resolution representation of that image. Performing a wavelet transform results in a coarse approximation comprised of *scaling coefficients*, and a series of detail coefficients, called *wavelet coefficients*. The scaling coefficients capture the low frequency, coarse content of the image and preserve its overall appearance. The wavelet coefficients capture the high frequency, fine details of the image and preserve its definition. Details occur in an image wherever there is a large difference in intensity between adjacent pixel values; that is, along a contour in the image.

Each successive level of a wavelet transform operates on the image by both first transforming all the rows, and then transforming all the columns, or vice-versa. This separable transformation produces an image whose first quadrant contains scaling coefficients, and whose remaining quadrants contain wavelet coefficients corresponding to contours of horizontal or vertical orientation. This partitioning is illustrated in Figure 1.

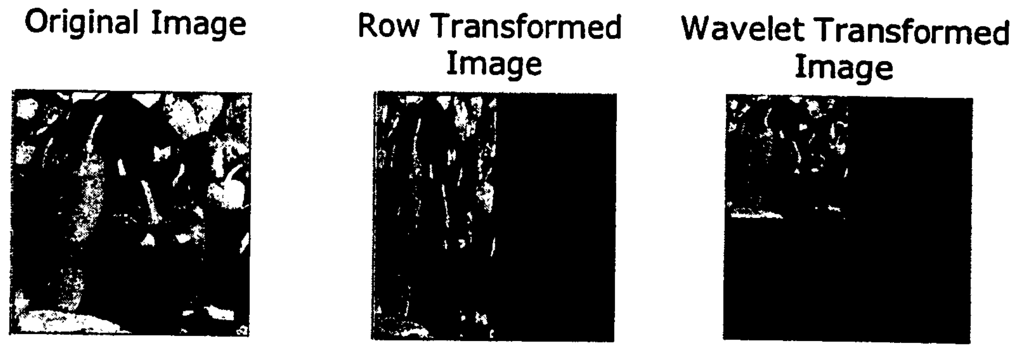


Figure 1 --- Row-and-Column Wavelet Transform

The wavelet transform is applied to an image multiple times by iterating the transform on the scaling coefficients. Each successive transform produces a set of wavelet coefficients for each level, along with a new coarse approximation. The wavelet coefficients at the lower scales correspond to fine image details, while wavelet coefficients at higher scales represent coarser image information. This yields a multi-resolution representation, as illustrated in Figure 2. In this figure, and throughout this work, the term "Mr-Level" is used as a shorthand for "Multiresolution Level."

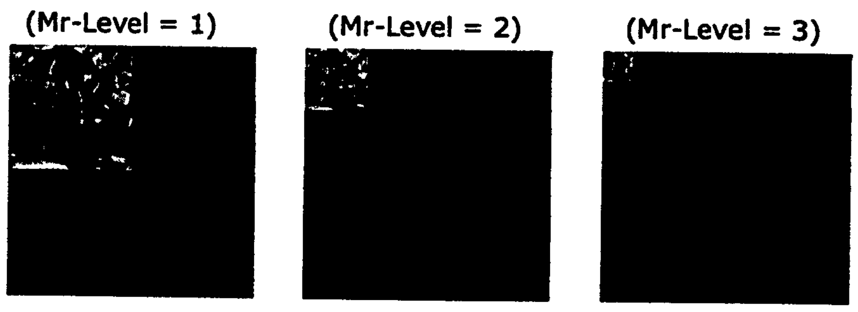


Figure 2 --- Multiresolution Wavelet Transform (Three Levels)

Evident in this figure is that, after three successive transforms, there is a large amount of redundancy that can be effectively compressed by a suitable encoding scheme, such as run-length encoding. This space appears to be purely empty in the above figures, but reversing the colors of the transformed image and enhancing the contrast shows that edge information is clearly preserved within it. This is illustrated below in Figure 3, where the upper-right quadrants of each transform level contain vertical edges, the lower-left quadrants contain horizontal edges, and the lower-right quadrants contain regions of overlap between edges of the two orientations.

The decomposition shown in Figure 3 contains three distinct regions of wavelet coefficients, each comprised of three quadrants preserving edges of the scaling coefficients visible in the upper-right. Each of these distinct regions is called a *sub-band*, with successive such sub-band containing finer details while occupying less area than the previous sub-band.

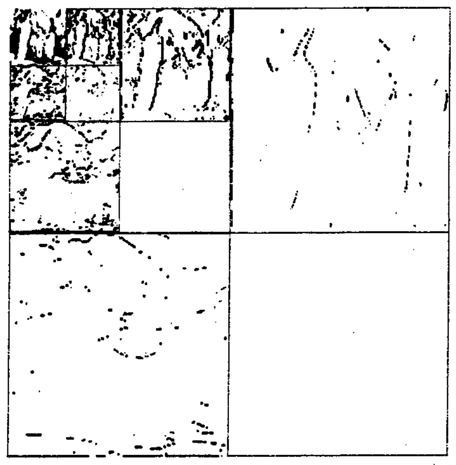


Figure 3 --- Three-Level Wavelet Transform Showing Edge Preservation

Thus, the largest areas of the transformed image (created in the first iteration of the transform) contain the least important data. Again, with a suitable encoding scheme, this redundancy is compressed.

The Haar* and Super-Efficient Haar* Wavelet Transforms

During this study, two versions of a modified wavelet transform are employed. The transform used is an integer version of the simple Haar Wavelet, hereafter referred to as the Haar* Wavelet. The Haar* wavelet filter is shown graphically in Figure 4.

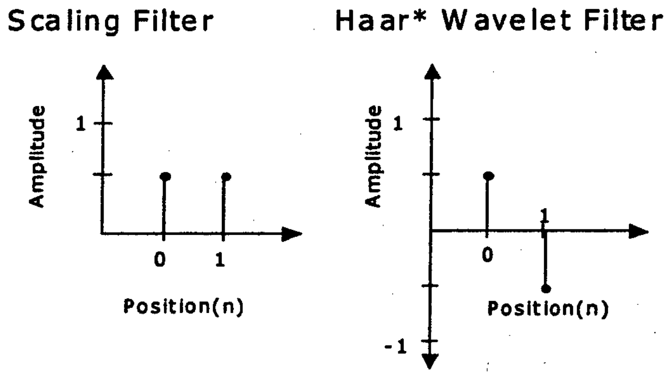


Figure 4 --- The Haar* Wavelet System

This filter differs from the non-integer Haar Wavelet filter in that the traditional wavelet filter has its amplitude scaled by a factor of $1/\sqrt{2}$. The Haar* wavelet filter is used because it permits integer rather than floating point arithmetic, simplifying and speeding computations, particularly in a hardware implementation. It is evident from Figure 4 that the Haar* wavelet transform is simply an averaging algorithm. Each scaling coefficient is the sum of two adjacent values, divided by two, and each wavelet coefficient is the difference of two adjacent values, divided by two. When adjacent pixels are close in value, the wavelet coefficient will be near zero. Only when the adjacent values differ significantly along contours in the image will the wavelet coefficient be of significant value. Each successive level of the transformation operates on the scaling coefficients generated in the previous level. The Haar* wavelet system may be described algebraically as

$$h_n = \begin{cases} \frac{1}{2} & \text{for n} = 0,1 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

and

$$g_n = \begin{cases} \frac{1}{2} & \text{for n} = 0 \\ -\frac{1}{2} & \text{for n} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where h_n is the scaling function and g_n is the Haar* wavelet. These equations serve as the basis for both wavelet transform designs presented in this work.

Standard (Non-Efficient) Design

The standard design of the Haar* Wavelet Transform, regardless of the specific implementation used (software, hardware, etc.) follows the flow of Figure 1. That is, the image is transformed first by rows, creating an intermediate image. This intermediate image is then transformed by columns, creating the final transformed image at that level. The intermediate image contains low frequency coefficients on the left and high frequency coefficients on the right. Both of these regions are further separated into high and low frequency regions. Figure 5 illustrates this process explicitly.

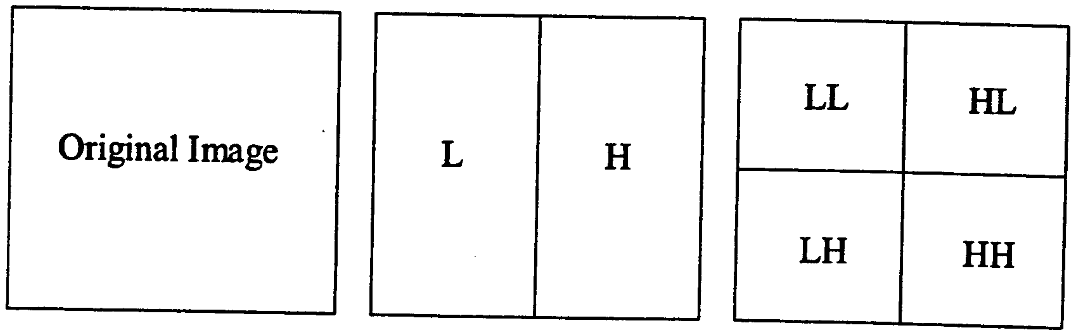


Figure 5 --- Illustration of Transform Frequency Regions

In the intermediate block of this figure, "L" represents a low frequency region and "H" represents a high frequency region. In the final block, separating the low frequency coefficients into high and low frequency regions produces the two quadrants on the left, labeled "LL" and "LH" for "low-then-low" and "low-then-high." Separating the high frequency coefficients in this same manner produces the two quadrants on the right, labeled "HL" and "HH" for "high-then-low" and "high-then-high."

When implementing this design, the intermediate image shown in Figure 5 is stored in a memory location separate from that containing the original image. This intermediate image is then transformed and the final image stored in yet another location, possibly overwriting the original image. The drawback to this approach is the cost incurred in writing the intermediate image, and immediately reading it back again. If the intermediate step could be eliminated, writing only the final image to memory, a considerable amount of time would be saved and the efficiency would be increased significantly. This is achieved by the Super-Efficient Haar* Transform.

Super-Efficient Design

The need for the intermediate row-transformed image occurring in the standard Haar* Transform can be eliminated through a fairly straightforward algebraic reduction of the equations used to compute the final transform. During the row transformation, pairs of adjacent pixels are filtered to create one scaling coefficient and one wavelet coefficient per pair. The column transformation could begin as soon as the first pixel pair on the second row is filtered. Rather than transforming the entire first row before beginning the column transformation, one could filter the first pixel pair on each of the first and second rows, and then filter the resulting coefficients to compute the column transformation for that portion of the image. Thus, by filtering 2x2 blocks of pixel values, the final transformed coefficients can be directly determined without first computing an intermediate image. This process is illustrated in Figure 6.

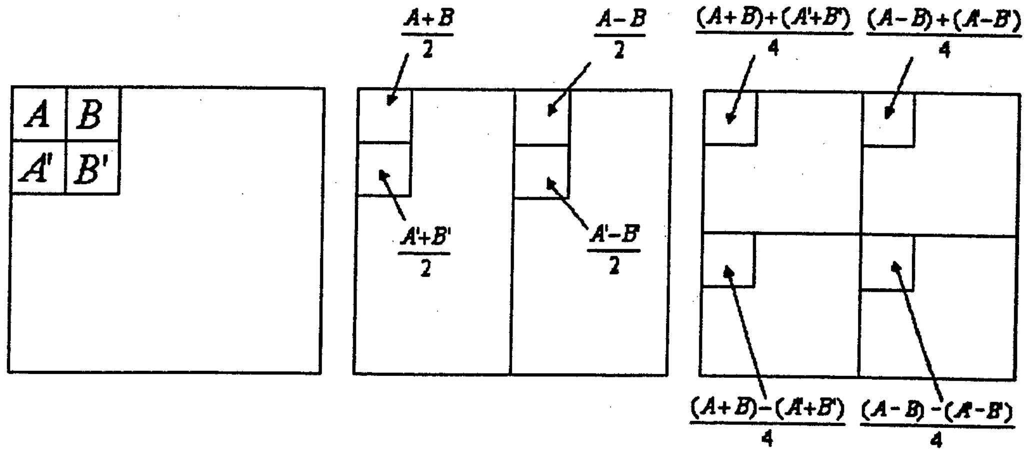

A
B
A'
B'
$\frac{A+B}{2}$
$\frac{A-B}{2}$
$\frac{A'+B'}{2}$
$\frac{A'-B'}{2}$
$\frac{(A+B)+(A'+B')}{4}$
$\frac{(A-B)+(A'-B')}{4}$
$\frac{(A+B)-(A'+B')}{4}$
$\frac{(A-B)-(A'-B')}{4}$

Figure 6 --- Relationship Between Intermediate and Final Transformed Images

In the above figure, the first pixel pair is represented by the symbols *A* and *B*, and the second pixel pair is represented by the symbols *A'* and *B'*. Applying (1) and (2) to these values produces the row-transformed scaling and wavelet coefficients, respectively, shown on the first row of the intermediate image in Figure 6. Applying these same equations to the second pixel pair produces the row-transformed scaling and wavelet coefficients, respectively, shown on second row of the intermediate image. These coefficients now provide two new column pixel pairs. Applying the scaling and wavelet filter equations (1) and (2) to these new pairs produces the fully transformed coefficients shown in the final image in Figure 6.

The algebraic relationship between the original pixel values and the final transformed coefficients is derived in Figure 6. The division by 4 in the final coefficient values results from finding the averages and differential averages of the intermediate coefficient values, all of which have previously been divided by 2. The block of four original pixels is now transformed by finding sums and differences of the original pixel pairs, and then adding and subtracting these sums and differences, dividing all final results by 4. Transforming all such 2x2 pixel blocks in this manner produces a fully transformed image.

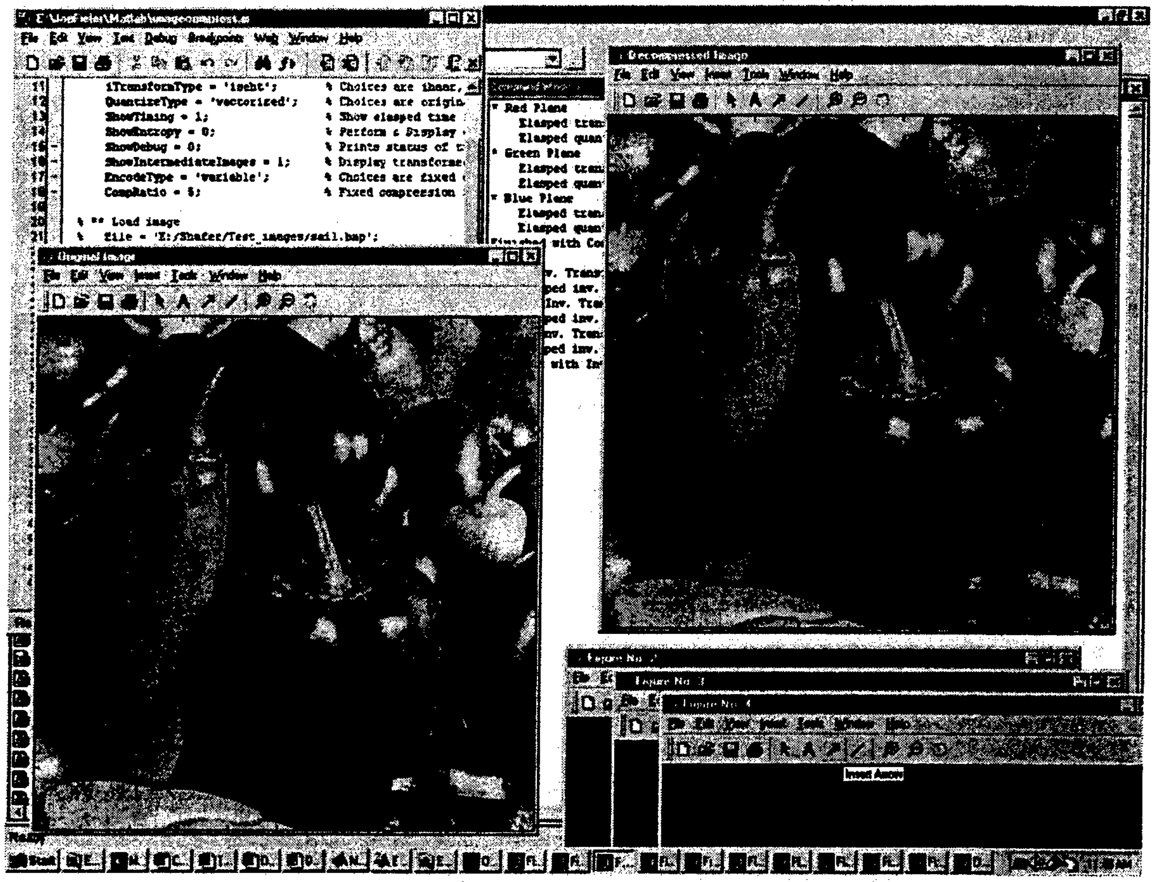
The implications of this Super-Efficient approach are significant. When implementing the standard Haar* in either software or hardware, the intermediate image must be stored to memory following the row transformation, only to be read back for use in the column transformation. By combining these two transformations into one, the number of read and write operations performed in the transform is reduced by half, resulting in much greater efficiency and faster operation. Combining the transformations algebraically rather than through parallel operations in software or hardware results in a much simpler and more easily implemented algorithm.

Integrated Test Environment

The software tool used to test, refine, and implement the wavelet transformation algorithms is Matlab. Matlab provides several built-in image functions that allow the user to quickly implement wavelet transform algorithms and display the results. Using another program tool such as C++ requires extensive time programming the system to do simple tasks such as reading and writing a bitmap, or displaying the transformed images.

The Matlab programs are written in a modular fashion so that various sections of the transform can be easily replaced with other transforms. This method allows for rapid testing of different algorithms. The Haar* transform can easily be replaced with a more efficient version of the transform, such as the Super-Efficient Haar* Transform (SEHT), or a completely different transform like the TS wavelet transform. Other sections of the compression scheme, like the encoder, can be replaced in a similar manner. The end result is that new ideas can easily be incorporated in the system without a major reworking of the rest of the system.

Finally, Matlab is used as a front-end interface with the Slaac1-V board. The Matlab program displays any bitmap images, calls the C++ program that initializes the Slaac1-V board, and reads the output obtained from the Slaac1-V board. Lastly, Matlab displays the end result. Matlab executes much slower than other languages like C++, but it is much easier to use in a development capacity. Below is a screen capture of the actual Matlab test environment.



```
     iTransformType = 'iseht';            % Choices are ihaar,
     QuantizeType = 'vectorized';         % Choices are origin
     ShowTiming = 1;                      % Show elapsed time
     ShowEntropy = 0;                     % Perform & Display
     ShowDebug = 0;                       % Prints status of t
     ShowIntermediateImages = 1;          % Display transforme
     EncodeType = 'variable';             % Choices are fixed
     CompRatio = 5;                       % Fixed compression

  % ** Load image
  %   file = 'E:/Shafer/Test_images/sail.bmp';
```

```
* Red Plane
    Elapsed tran
    Elapsed quan
* Green Plane
    Elapsed tran
    Elapsed quan
* Blue Plane
    Elapsed tran
    Elapsed quan
```

Figure 7 – The Matlab Integrated Test Environment

Test Results

The first bit of information to observe is the improvements made from the Haar* Transform to the Super Efficient Haar* Transform. The following figure depicts these results.

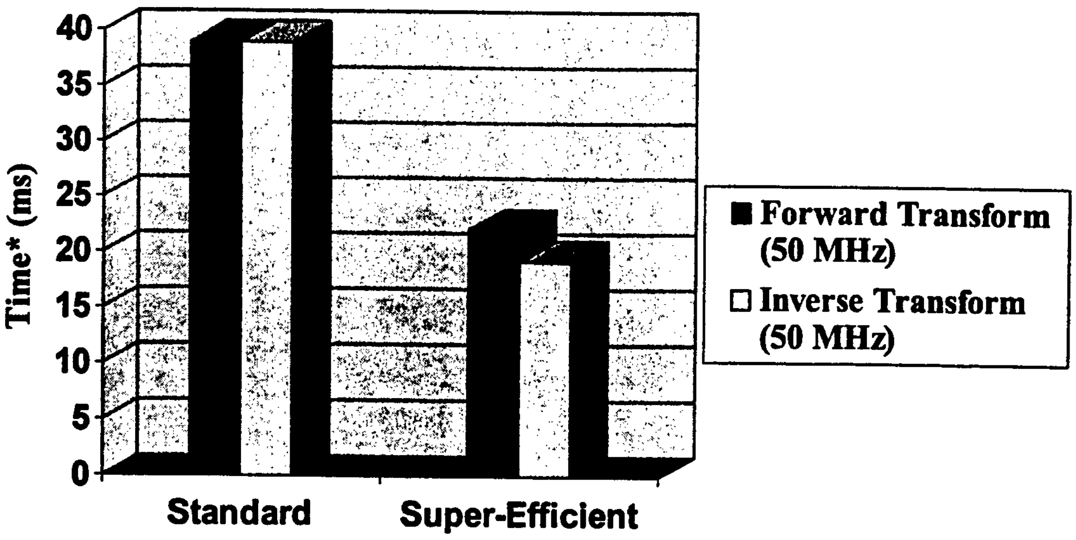


Figure 8 – Improvements made to the Haar* vs. Super-Efficient Haar* (512x480 Image)

The next important set of data is found when the entire compression and decompression systems are finalized and simulated. The following figure shows a graph illustrating the results of the testing.

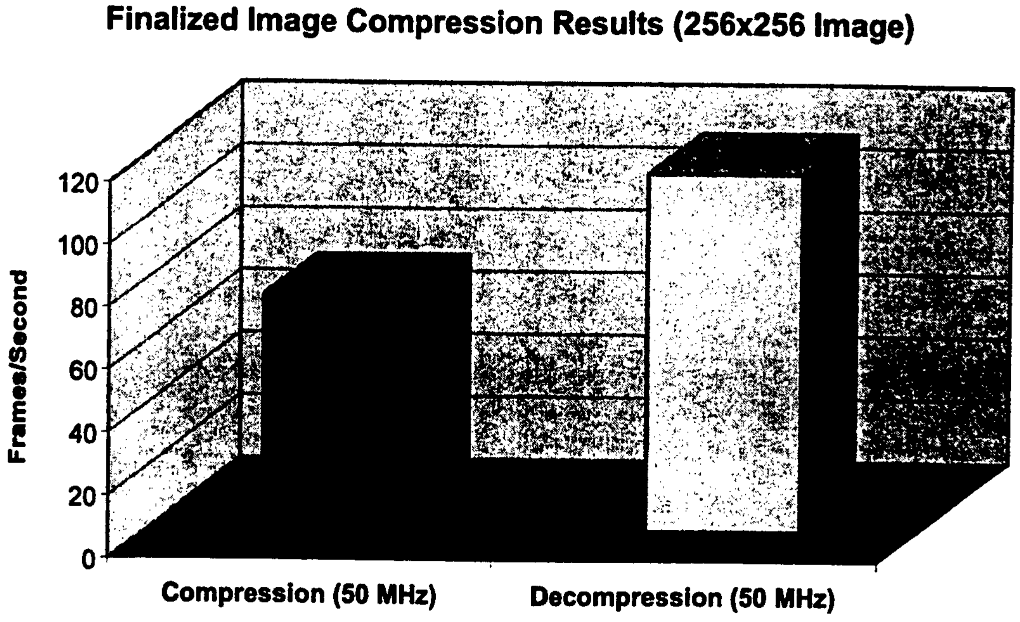


Figure 9 --- Finalized Image Compression Results (256x256 Image)

It is noted here that the human eye's threshold to clearly differentiate between images occurs at approximately 30 frames/second. Therefore, at rates higher than 30 frames/second, most humans will not notice individual frames and will see "smooth" video.

The main focus of research on this project is the wavelet transform, but some time was devoted to an encoder and decoder so that a complete compression system could be built. Both lossless and lossy run-length encoding schemes were implemented. Run-length encoding looks for adjacent groups of identical pixels. Figure 10 shows the order that the run-length encoding is implemented.

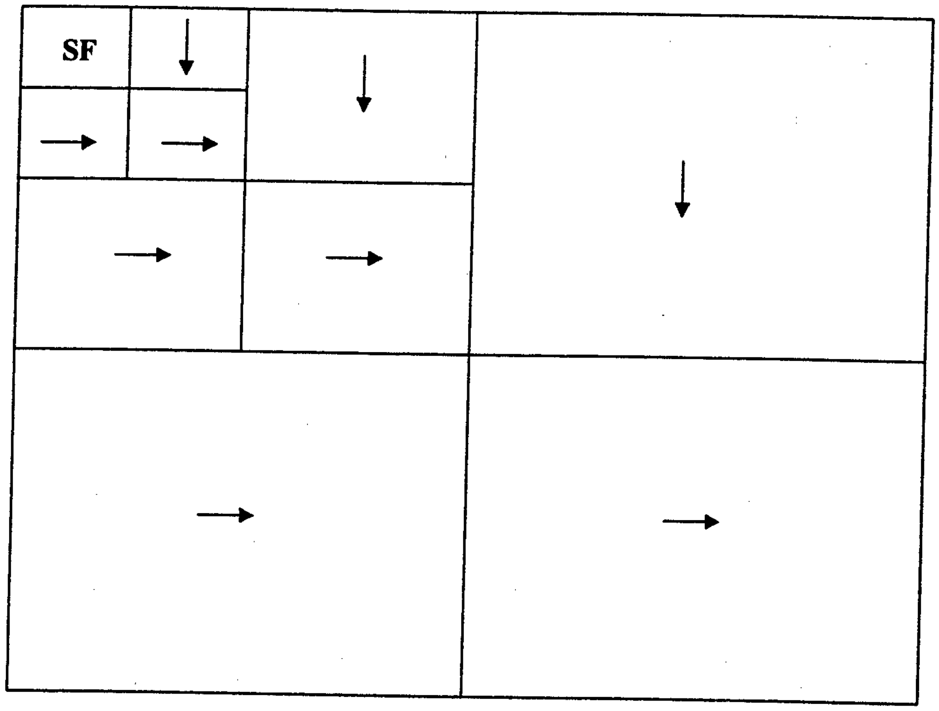


Figure 10 – The Run-Length Encoding Scheme

The scalar section is not encoded because it has very few pixels that are similar. The direction of the run-length encoding follows the arrows. This method is done because the sections with the vertical arrows usually only read the horizontal edges of the image, and the horizontal arrows occur in sections where the sections only read the vertical edges in the image. Therefore, it is possible to obtain long runs of numbers.

The lossless run-length encoding will go through the image one section at a time and make sure every bit of data is encoded. The decoded image will be exactly the same as the encoded image. The lossy run-length encoding that is implemented is just a modified version of the lossless encoding. The user specifies a specific compression ratio, and the hardware will encode the image until the specified size is reaches. Normally, the image stops encoding in the third sub band. The third sub band contains little information about the image so there is little loss in image quality if this region is excluded.

Conclusion

The main focus of this research is to implement a fast, effective, and inexpensive image compression scheme, in hardware, through the use of the Super-Efficient Haar* Wavelet Transform. The results obtained are comparable to industry standard. Moreover, with additional funding several improvements could be made to enhance the overall compression ration and/or compression time. For instance, a more proficient encoding scheme could be utilized to improve performance. The encoder is just one example of the numerous improvements that could be made. In closing, it is important to realize the goal of the project, which is to create a cheap, useful hardware compression scheme. Clearly, all the objectives have been accomplished.

<u>*TASK 22*</u>

Fast Mixed-Energy Simulation in a Distributed Environment

Principal Investigator:	Prof. Harold W. Carter
Institution:	University of Cincinnati, Cincinnati, Ohio
Period of Performance:	July 1, 2001 to June 10, 2002
Task:	The objective of this research is to discover, study, implement, and analyze methods for simulating large behavioral models of mixed-energy systems (i.e., electronic circuits, mechanical components, microwave systems, etc.) with significantly increased speed over that available today. Further, investigation into efficient methods for applying design of experiments and statistical methods for efficiently designing and executing user-defined multiple simulation studies. The result of this effort is to be a robust, fast VHDL-AMS simulator capable of executing on uniprocessor and distributed processor platforms and a GUI-based simulation planning, execution, and analysis environment for conducting multi-simulation studies.

Advanced Continuoue-Time Methods for VHDL-AMS Mixed-Signal Simulation

Sanjiv Pande
Shishir Agrawal
Vikram Krishnamachary
Sachin Bapat
Geeta Balakrishnan
Harold Carter

University of Cincinnati
Cincinnati, Ohio

July 2002

1 Introduction

This report documents the efforts of a team of five student researchers in creating a VHDL-AMS simulator called Sierra that is complient with the IEEE 1076.1 standard. Building upon the SAVANT VHDL simulator created by Prof. Phil Wilsey and his students at the University of Cincinnati, algorithms unique to the continuous-time and mixed-time features of VHDL-AMS were added by the team. Several approaches to improving the speed of analog simulation were incorporated: 1) simple simultaneous equation reduction, and 2) incremental matrix build. Finally, the simulator was incorporated in a experiment design and visualization system to permit the design, execution, and analysis of multi-factor experiments to aid model optimization.
It was the desire of the team to exploit and build upon the parallel execution features both of SAVANT to produce a parallel VHDL-AMS system. Alas, the version of SAVANT (summer 2001) that was used did not have a complete parallel implementation and thus the AMS extensions describe here were sequential in nature only. An effort has recently started to extend the Sierra simulator to be parallel by using the very latest (fall 2002) SAVANT simulator that executes in parallel. That effort will be documented upon its completion around summer 2003.
This report is divided into the following sections:

1. Introduction
2. Overview of the Sierra Architecture
3. Elaboration
4. The Analog Kernel
5. Improving Simulation Execution Speed
   (a) An Equation Reduction Technique for VHDL-AMS Simulation
   (b) Reducing the Matrix Build Time

2 An Overiew of the Sierra Architecture

The Sierra VHDL-AMS simulator is a compiled simulator where execution consists of three basic tasks: 1) Analysis of the input VHDL-AMS models to produce C++ files representing the model-specific semantics of the input models, 2) C++ compilation of the C++ files and linking the resulting object files with the precompiled kernel to produce a executable simulator, and 3) execution of the simulator. Figure 1 shows these three general tasks as a part of the overall architecture of Sierra. The various modules of the mixed-signal simulator include an analyzer, code generator, elaboration kernel, digital and continuous time kernels. These modules are described below in detail.

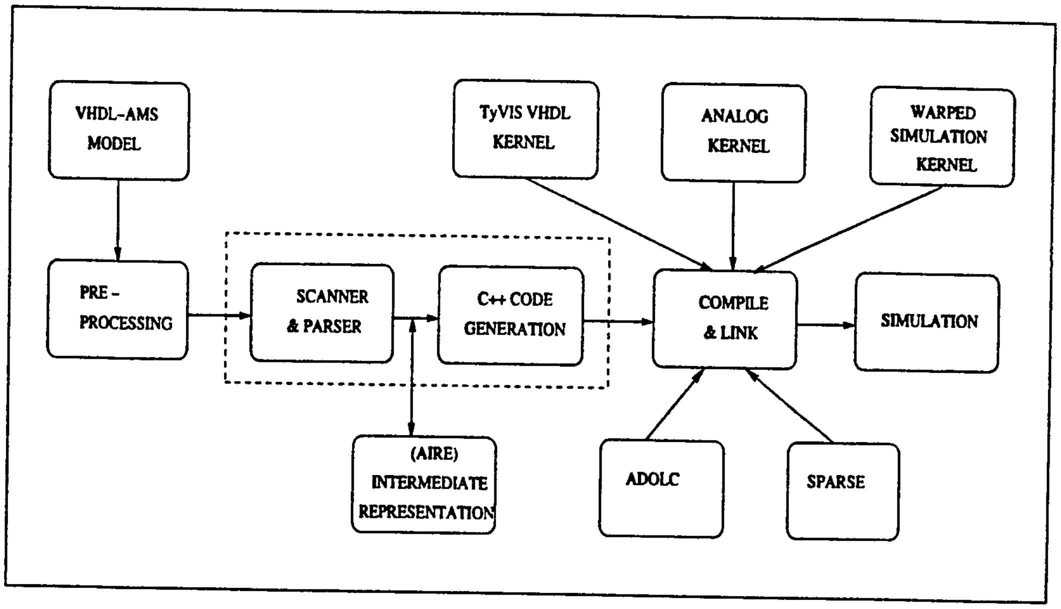


Figure 1: Sierra Architecture

2.1 Preprocessor

There are a few VHDL-AMD constructs like case-use and like multiple 'dot attributes associated with any quantity that cannot be handled in the back-end. However, these constructs can be simulated if we can transform these statements in the input file into something which the front end parser and back end kernel can handle. So we can still simulate them. The preprocessor does exactly this task. The input VHDL-AMS file is first preprocessed to generate an equivalent VHDL-AMS description. Thus the case-use statement is transformed into an equivalent if-use statement. For quantities with multiple 'dot attributes, additional quantities are introduced such that every quantity has a single 'dot. Thus the introduction of this preprocessor stage helps in simplifying the parser and back end kernel. A set of Perl scripts are used to preprocess the input VHDL-AMS description. The output of this stage is the input to the VHDL-AMS analyzer.

2.2 VHDL-AMS Analyzer

Though any VHDL-AMS description can be manually translated to C++ code compliant with the simulation kernels, the process is tedious and time consuming even for small models. This process is automated using a scanner, parser and code generator. The Purdue Compiler Construction Tool Set (PCCTS) [32] was used to develop such a parser for VHDL-93 at the University of Cincinnati. We extended the SAVANT[1] [37] analyzer to support the parsing and analysis of VHDL-AMS descriptions. The analyzer checks the syntactic and semantic correctness of the input description and generates the intermediate respresentation of VHDL-AMS. The intermediate representation generated follows the standard Advanced Intermediate Representation with Extensibility (AIRE) [36].

2.3 Intermediate Representation

The AIRE specification addresses the simulation language user's need for efficient mechanisms for sharing design information between tool components. The AIRE specifications includes coordinated internal intermediate representation (IIR) and file intermediate representation (FIR) specifications. AIRE targets tools beginning early in the development cycle or at a major system redesign. AIRE delivers on key user requirements including performance, capacity, portability, extensibility and reusability [36]. Figure 2 shows the IIR hierarchy for implementing Quantity Declaration. As can

[1]Standard Analyzer for VHDL Applications for Next Generation Technology

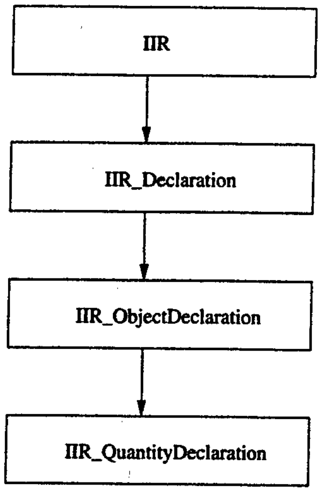


Figure 2: IIR class hierarchy for Quantity Declaration [36].

be seen from this figure, to add support for Quantity declarations in VHDL-AMS we extend the existing IIR_ObjectDeclaration class as quantities are the new objects in AMS. Thus we are able to use the existing features provided by the base class and add new features specific to quantities in the leaf-level class.

2.4 VHDL Kernel

The TyVIS kernel (the VHDL discrete-event simulation kernel in SAVANT) requires a mapping from a VHDL model to a C++ representation that can be simulated. A TyVIS kernel extended to support VHDL-AMS is used. This extended kernel performs the elaboration of the design, creation of DAE objects, propagation of signals, their resolution and type conversion, file I/O, and other such requirements of VHDL-AMS. The simulation of discrete-event models is accomplished using WARPEDa system, an optimistic Time Warped simulator core (also a part of the SAVANT system) developed at the University of Cincinnati.

2.5 Code Generator

The code generator has been built as an extension to the AIRED VHDL intermediate representation. The structure specified in the AIRE is implemented as C++ classes and their public interfaces as public member methods. C++ class inheritance is used to clearly separate the AIRE specifications from the user defined functions and the analyzer. The parser creates the parse tree, instantiating the AIRE objects to preserve the structure and information of the input VHDL-AMS model. The code generator publishes the C++ equivalent of the VHDL-AMS descriptions. The published code is compliant with the simulation kernels, and when compiled with the kernels and library components, a simulation executable is obtained. This executable when invoked, performs run time elaboration and performs the simulation. Besides C++ code generator, a VHDL-AMS code generator is also present. This regenerates VHDL-AMS code that is functionally equivalent from the parse tree created by the analyzer.

2.6 Elaboration

Before a VHDL-AMS description can be simulated, it must first be elaborated. The language reference manual defines elaboration as the process by which a declaration achieves its effect [18]. The process of flattening out the hierarchy of the design to obtain a collection of processes, signals, netlists, characteristic expressions representing the DAEs, and quantities is termed as elaboration

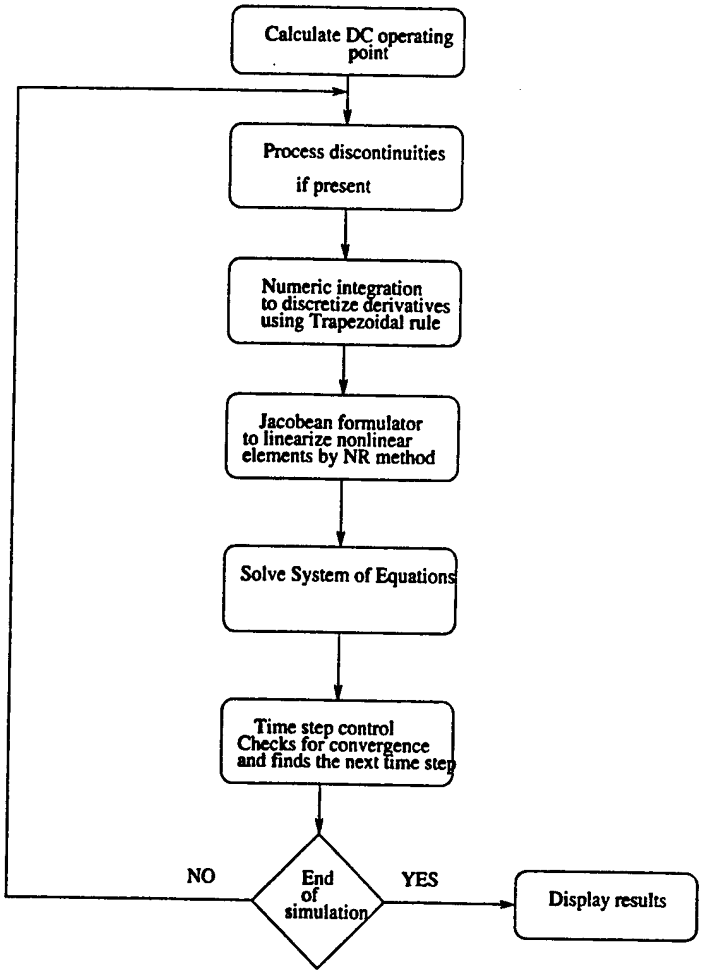

Calculate DC operating point
Process discontinuities if present
Numeric integration to discretize derivatives using Trapezoidal rule
Jacobean formulator to linearize nonlinear elements by NR method
Solve System of Equations
Time step control Checks for convergence and finds the next time step
NO
End of simulation
YES
Display results

Figure 3: Analog simulation flow and various routines used for simulation

of the design hierarchy. Sierra uses the elaboration strategy called RTEMS (run time elaboration for VHDL-AMS) defined by Karthiresan [30].

2.7 Continuous-time kernel

The elaboration produces a continuous process. The continuous process finds the solution of the DAEs contained in the process at specific time points determined by the time step algorithm. The core of the continuous process is the continuous time kernel or the analog kernel. The analog kernel in Sierra is essentially derived from SPICE. It uses the direct method as is used in SPICE. The continuous time process includes modules to perform automatic time step control, numeric integration, Jacobian formulation and synchronization routines. The DAE represented as equation objects and the varibles referred as quantity objects are present in this continous process. The details of the various modules that form a part of this Continuous time kernel is presented below. Figure 3 gives the flow for analog simulation and identifies where the difference blocks explained below fit during the simulation cycle.

Equations and quantities Equations represent the interaction between the variables in the system. The variables are known as *quantities* in VHDL-AMS. Objects are used to abstract the equation and the quantities in the system in the generated C++ code. The simulation kernel uses these equation and quantity objects for performing simulation. Hence, all the information including the terminal associations, attributes, conservative semantics, initial values, and tolerances, as described in the input VHDL-AMS model are preserved in the modular representations.

Numeric Integration The DAE system is integrated to converted it into an algebraic equation system. Numerical integration methods are used to perform integration. There are two broad classes of integration methods: *explicit* and *implicit*. There are several known methods for numerical integration [27], and the choice of a particular method influences the balance between speed, accuracy and the ability to converge in a simulator. Basically, implicit methods are preferred because they exhibit better stability and accuracy. Sierra uses the *Trapezoidal* method of integration. Trapezoidal method is an implicit integration method. Consider a differential equation of the form

$$x' = f(x,t) \tag{1}$$

where x' is the derivative of x with respect to time. The trapezoidal integration applied to the above equation results in:

$$x_{n+1} = x_n + \frac{h}{2}[x_n' + x_{n+1}'] \tag{2}$$

where (n+1) is the current iteration and h refers to the current time-step. The trapezoidal algorithm is a two-step formula because the values of x' at two instants of time, t_n and t_{n+1} are needed. Since it uses the derivative at the past and current time-point, trapezoidal method tends to be more accurate than other integration methods [23].

Jacobian formulator The linearization of a non-linear system of equations which results in a linear system of equations and to apply linear solution methods, the partial derivatives of equations with respect to all the variables is required. This is called as the *Newton-Rahpson* method for linearizing the equations. The matrix or the partial derivatives is called as *Jacobian* matrix. This is the A matrix used in finding the solution. Determining the Jacobian analytically is not possible for all DAEs. Sierra uses the ADOL-C automatic differentiation package [16]. This package facilitates the evaluation of first and higher derivatives of functions that are defined by computer programs written in C or C++. As will be discussed in Chapter 4, this phase is expensive and we propose a approach to perform this only when its necessary.

Timestep control The timestep control algorithm used in Sierra is adopted from SPICE [23] and determines the time points where the DAEs are solved. The timestep algorithm, varies the step size in the following two ways:

- When a solution failst to converge for a timestep, the timestep is decreased by a factor of eight and solution is attempted. Non-convergence occurs when the values of quantities in VHDL-AMS descriptions change at a rapid rate. The reduction of the timestep is required to reduce the probability of non-convergence and to maintain solution accuracy over the time-step.
- The time-step is increased during periods of little or no activity to speedup simulation. This is accomplished by increasing the time-step by two, if at the previous timestep convergence was obtained.

The time-step control method has upper and lower limits on the size of the time-step, and timestep taken must be within these limits.

Discontinuity processing Discontinuity processing needs to be done during simulation only if needed. Discontinuity is an abrupt change in the values of a continuous waveform. Such discrete changes within continuous systems is very common. Discontinuities occuring during simulation of continuous or mixed-signal systems either cause numerical algorithms to fail or give rise to significant errors. Discontinuites may occur in mixed mode simulation due to the followin reasons:

- Dynamically changing equation sets.
- Interaction between discrete and continuous variables

- Discrete changes in the primary inputs of an analog subsystem.

Break statements model discontinuities in VHDL-AMS. Discontinuities must be correctly detected, located and handled appropriately for valid simulation. A detailed description on handling discontinuities in a mixed-signal simulator for VHDL-AMS can be found in [28].

2.8 New Features in Sierra

This section describes the new features that were added to Sierra as part of this research work. These features are new additions; they were not incorporated in SEAMS[13], the initial VHDL-AMS simulator created at the University of Cincinnati in 1998.

2.8.1 Array Natures

An array nature is a composite nature. Composite natures are used to define collections of terminals. These include array of terminals and records of terminals. Terminal objects of an array nature consist of identical elements, each associated with some index value. The branch types defined by an array nature definition are array types. A terminal object of an array nature is referred to as an array object, just as is an object of an array type. Array natures can be unconstrained or constrained [18]. In this research unconstrained array natures were implemented. Here is an example of a one-dimensional unconstrained array nature:

```
nature Electrical_vector is
                array (NATURAL range <>) of Electrical;
```

Here *Electrical* is a scalar nature as defined in Figure 6. Now *Electrical_vector* is an array nature and can be used to define an array of terminals. We cannot create an object of an unconstrained nature, hence we declare a subnature as follows:

```
subnature Electrical_bus is Electrical_vector( 1 to 8);
```

A subnature puts a constrain on an unconstrained nature. The above defines a subnature which restricts the Electrical vector to size eight whose individual elements have index ranging from 1 to 8. Now we can create an object or a terminal of this subnature. Addding support for this construct involved modifying the parser and code generator. Below we mention the steps involved in implementing Array Natures:

- **Parser:** We need to add proper grammar rules for parsing array nature definitions. The grammar is defined in the language reference manual [18]. After properly matching the grammar rule, we create an intermediate form for this declaration. This corresponds to creating an object of IIR_ArrayNatureDefinition class as defined in AIRE [36]. Figure 4 shows the details of this IIR_ArrayNatureDefinition class.

- **Code generation:** In the generated code the array nature definition appears as an object of class *ArrayTypeInfo*. Figure 5 shows some features (i.e., methods) of this class. This class is a placeholder for storing information about arrays described in the input VHDL-AMS file. For the example of array nature definition shown above, the following code is generated:

  ```
  ArrayTypeInfo Electrical_vector; ArrayTypeInfo Electrical_bus;
  ```

  Thus during elaboration, proper object will be created which will correspond to the characteristics of the array nature defined by the modeler. Since this is just a type, its not simulated. However, it is used for identifying the nature of the terminals.

The steps involved here are similar to Array Terminals but involve modifications to the analog kernel as Quantities are the objects that appear in simultaneous equations and are the unknowns in the system. The steps involved in implementing array and indexed quantities are listed below:

IIR_ArrayNatureDefinition

This predefined class represents natures containing zero
or more instances of the same elemental subtype
Public Methods

Constructor Method

The constructor method creates a valid array type definition
IIR_ArrayNatureDefinition();

Index Subtype Methods

The index subtype methods refer to the array's index domain
*void set_index_subtype(**IIR_ScalarTypeDefinition*** *
index_subtype);
IIR_ScalarTypeDefinition * *get_index_subtype();*

Element Subtype Methods

The element subtype methods refer to the element subtype
which is replicated to form the array.
*void set_element_subtype(**IIR_NatureDefinition*** *
element_subtype);
IIR_NatureDefinition * *get_element_subtype();*

Figure 4: Description of IIR_ArrayNatureDefinition as defined in AIRE [36]

ArrayTypeInfo

This class holds info about the array type
TypeInfo class used in this class
is a general purpose class that acts
as a base class for all type infos
Methods

Constructor Method

```
ArrayType(int, TypeInfo*, ...);
```

Other general methods

```
TypeInfo* get_rangeInfo(const int) const;
int get_dimensions() const;
int get_number_of_elements(int dimension) const;
VHDLType* createObject(ObjectBase::ObjectType) const;
ArrayInfo getBounds(int) const;
```

Data members for this class

```
TypeInfo **ranges;
int noofDimensions;
TypeInfo *elementTypeInfo;
bool unconstrainedType;
```

Figure 5: Arraytype class.

- **Parser:** After the grammar rule corresponding to a quantity declaration is matched, we create an object of IIR_BranchQuantityDeclaration has to be created. This class is derived from the IIR_QuantityDeclaration class as shown in Figure 2. The LRM governs finding the proper across and through types of these quantities depending on the terminals. If both terminals are of a scalar nature, then the type is the across type implied by the plus terminal name. If only one terminal denoted by the terminal aspect is of a composite nature, then the type is the across type implied by that terminal name. If both terminals are of composite natures, then the type is the across type implied by the plus terminal name [18]. After determining the proper type of the quantity, appropriate intermediate information is stored.

- **Code generator and back-end kernel:** During code generation, an ArrayType object corresponding to this quantity is generated. For the above quantity declarations, the following code is generated:

  ArrayType vr, ir, vr1, ir1;

  In the back end kernel, proper constructors were introduced to handle quantities defined between array terminals. For indexed quantities, appearing in simultaneous equations, the generated code indexes the above array objects, which is achieved by properly overloading the '[]' operator in the back-end kernel. During elaboration checks have to be performed if the modeler has used an indexed quantity such that its index falls outside the allowed range for that array. Proper error is reported in that scenario.

2.8.2 Generate statement

All the features introduced above can be used to create large and scalable models easily by using generate statements. The generate statements are used to replicate some structure or to duplicate some units in some form of a pattern. The following example shows the use of generate statement:

```
for I in 1 to 10 generate
        begin
                res: vr(I) == ir(I) * 100.0 ;
        end
end generate ;
```

From the above we see that we can easily create ten resistors using a very simplified model description. Without the support for arrays in VHDL-AMS, the modeler would have to specify ten seperate simultaneous equations. The implementation of generate statement, involves generating a corresponding *for loop* in the C++ code. This then falls naturally into the way by which normal simultaneous statements are executed in the kernel.
A complete example with all the above features is given in Figure 6. The input VHDL-AMS description is much smaller than one without arrays. The addition of these capabilities to Sierra opens up the possibility of modeling bigger designs with a more structured modeling style.

3 Elaboration

A VHDL-AMS model must be elaborated before it can be simulated. After elaboration, initialization of signal nets and quantities takes place and then simulation proceeds. Simulation consists of repetitive execution of the simulation cycle as defined in [18]. The elaboration of a design hierarchy creates a collection of processes interconnected by nets and quantities whose values are defined by characteristic expressions(CEs). In a VHDL-AMS model design hierarchy can be defined by a design entity or a configuration. We limit our discussion to elaboration of design entities. A VHDL-AMS model is first analyzed for its syntax and semantics and then appropriately prepared for execution after code generation by SAVANT [37]. Elaboration takes place during the initial execution of the model and that is why it is called run-time elaboration. The elaboration methodology for a parallel VHDL simulation approach [52] contributes to elaboration of digital constructs in the model. Various phases of run-time elaboration are shown in Figure 7 and are explained below

Phase 1 - Construction of design units and objects In the first phase, the elaboration of a design hierarchy involves the *instantiation*, the *signal-net update* and the *connection*.

- **Instantiation -** The top most design entity that has been identified in the system serves as the input to this first step. In this step, creation of objects represented in the model *viz.* components and processes [2] are performed in top down approach. For each design unit, objects representing the declarations *viz-* types, variables, signals, subprograms, quantities, and terminals are created. Elaboration of a process statement involves the instantiation of the object corresponding to the process statement. At the end of this step, the set of all objects that are present in the design hierarchy is created.

- **Signal-net update -** In this step all information relating to signals in the system is recorded. It includes updating fanout and creating drivers for each of the signals, and association of processes with the drivers of each signal. This step takes place as a part of a design unit construction in a bottom-up fashion.

- **Connection -** This phase passes the information about the signals on to the instantiated components and processes to record the necessary data such as a signal source tree (for multiple drivers of signals) and up-down type conversion functions specified in the model.

This completes phase 1 of elaboration for a mixed-signal system.

Phase 2 - Construction of continuous system This phase identifies the different unknown quantities in the design and forms the characteristic expressions in order to accurately simulate the behavior of the continuous system. Thus, this phase serves as the construction phase for the continuous system. In this phase, the design in processed in four steps as follows:

[2]Process in this context represents the user defined VHDL processes which exhibit discrete-event behavior.

```
PACKAGE electricalSystem IS
        NATURE Electrical IS real ACROSS real THROUGH
        Ground reference;
        NATURE Electrical_vector is array (natural range <>) of electrical;
        SUBNATURE Electrical_bus is Electrical_vector(1 to 8) ;
END PACKAGE electricalSystem;
use work.electricalsystem.all;

-entity declaration

ENTITY array-model IS
END array-model;

-architecture declaration

ARCHITECTURE behavior OF array-model IS
        terminal T1, T2 : electrical; - scalar terminal
        terminal T3, T4 : Electrical_bus; - array terminal
        quantity vs across is through T1;
        quantity vr1 across ir1 through T1 to T3;
        quantity vr2 across ir2 through T3 to T4;
        quantity vr3 across ir3 through T3 to T2;
        quantity vr4 across ir4 through T4;
        constant R1 : REAL := 100.0;
        constant R2 : REAL := 200.0;

        BEGIN
        for I in 1 to 8 generate
        begin
           BE1: vr1(I) == ir1(I) * R1;
           BE2: vr2(I) == ir2(I) * R2;
           BE3: vr3(I) == ir3(I) * R1;
        end
        end generate ;
           vsrc : vs == 5.0
        END ARCHITECTURE behavior;
```

Figure 6: VHDL-AMS description showing modeling using array natures, terminals and indexed quantities

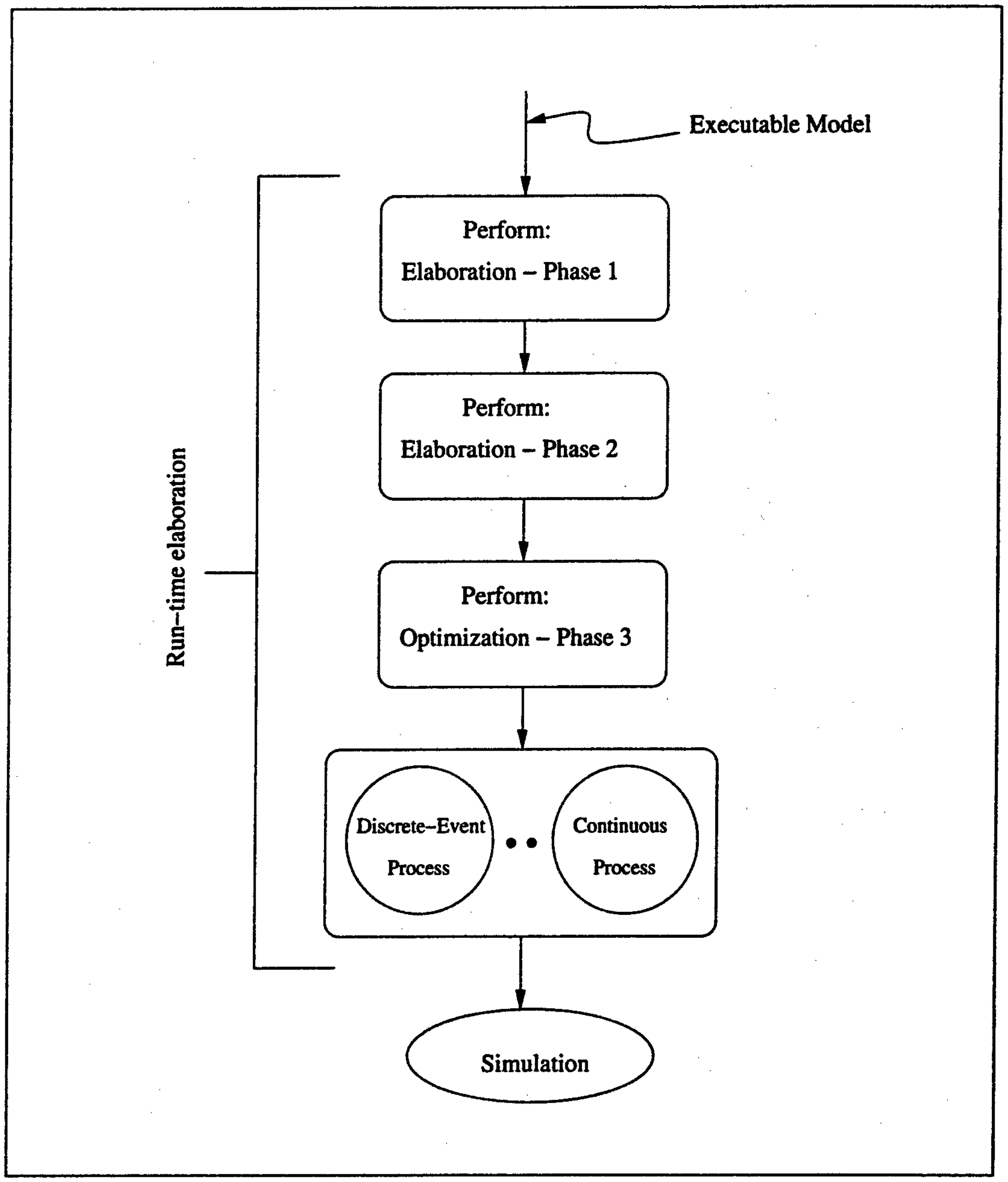


Figure 7: Phases of run-time elaboration of a design hierarchy

- In the first step, the number of unknown quantities are identified across the design entities. Solvabitiy checks for each external block are applied so that necessary conditions for solving the system of unknowns are satisfied. These checks are applied in the following manner:
  - Determine the *characteristic number*[18] of each external block in the design hierarchy.
  - Perform assertions as to determine the equality of the characteristic number of each block with the number of characteristic expressions formed for that block. These assertions are made after the last step of this phase.

  The analog solver determines the solution values for the unknowns during simulation.
- In the second step, the formal terminal and the quantity interface objects across the design entities are associated with the actuals in the top-down fashion beginning with the top-level design entity. This step is very similar to the *connection* step in phase 1. The following rule as stated in [18] must be satisfied to realize the semantics of interface quantities and terminals:
  - Interface quantities with mode out can be left unassociated.
  - The *nature* of the formal and actual objects must be the same.
  - If none of the formals in the set of quantities identified in the system has mode out then a *characteristic value* must be provided in the declarative region where the free quantity of unassociated interface quantity has been declared.
- In the next step, all the *break statements* in the design are processed to identify discontinuity augmentation set in the system. This is performed to accurately detect and process discontinuities in a mixed-signal system during simulation [28]
- Finally, the CEs are formed from the simultaneous statements as per the rules that govern their formulation [18]. CEs are the representation of ODAEs in order to evaluate the behavior of the continuous system. A few of the rules to form CEs are:
  - An explicit CE is formed for each simultaneous statement in the model.
  - An implicit CE is formed for each differential quantity that occurred in the simultaneous statements.
  - The value of each across quantity minus the reference quantity of its plus terminal plus the reference quantity of its minus terminal is a characteristic expression in the model.

  Since VHDL-AMS allows conditional simultaneous equations in the form of *simultaneous-if* and *simultaneous-case* statements, the formation of CEs maintains the control structure of these statements.

3.1 Elaboration of AMS constructs

3.1.1 Entity declaration

An entity declaration in VHDL-AMS defines an interface to a component. Every entity declaration is represented as a C++ class, deriving from the predefined *elaboration kernel* class, in the generated code. The elaboration kernel has generic routines which are useful for filling information for the design entities. This predefined class provides methods to create the net-list for the signals, collect information regarding the drivers of the signals, form the different characteristic expressions as recognized by the analog simulation kernel and obtain the hierarchical information such as the type conversion functions. The generic constants and the signals in the port list of the entity become member objects in the entity class. The quantities and terminals are generated as data members of each such design entity class. The signals however, are not instantiated as signals, but as a `SignalNetInfo` object which contains data structures to store the net list and information about the drivers of the signal.

3.1.2 Architecture declaration

An architecture body contains the concurrent and simultaneous statements that implement the interface specified by the corresponding entity. Elaboration of an architecture involves elaboration of its declarative region and the concurrent and simultaneous statements in its body. The concurrent statements are elaborated during the instantiate phase of elaboration. The elaboration of the declarative region is essentially the elaboration of the signals, quantities and terminals declared in the architecture. The net list and drivers of signals are created recursively, each concurrent statement creating drivers that it contains. Only the drivers in the process statements are created in the net-info phase of the elaboration. The information of the signals gathered by the concurrent statements are passed on to the higher level from where the corresponding entity was instantiated. Elaboration of an architecture declaration is implemented as a class with data elements corresponding to the signal, quantity and terminal declarations in the declarative region and methods to achieve the processing required in Phase 1 and Phase 2. This class is derived from the corresponding entity class, and hence inherits the data structured for the signals, quantities, and terminals in the port list of the entity. The default port associations of component instantiations are done here, which can be changed by a configuration declaration. The bounded entities of the entity declarations and component declarations instantiated in this architecture are identified and assigned during the instantiate step in Phase 1. As the elaboration of the concurrent statements occurring in an architecture have been discussed in [52] and also, as the focus is on AMS part of Sierra, we limit our discussion to the elaboration of the simultaneous statements, quantity attributes and break statements.

3.1.3 Elaboration of simultaneous statements

Elaboration of simultaneous statements results in the formation of the characteristic expressions. The characteristic expressions in Sierra are the different kinds of ODAEs recognized by the analog simulation kernel[10]. Hence the methods which form the CEs in the design entities must create them in a format recognized by the simulation kernel. The different ODAEs as recognized by the Sierra analog simulation kernel are as follows:

- **Across equation** - In the equation, $y = f(x)$, if y is an across quantity (*e.g.* voltage), then the equation is classified as an across equation.
- **Through equation** - In the equation, $y = f(x)$, if y is a through quantity (*e.g.* current), then the equation is classified as a through equation.
- **Free equation** - In the equation, $y = f(x)$, if y is a free quantity (*e.g.* distance), then the equation is classified as free equation.

The C++ code generated to perform the elaboration of the simultaneous statements create the above mentioned *characteristic expressions* from the information available in the AIRE intermediate format. All the unknowns that occur in a simultaneous statement are part of the CE formed for that particular statement. Since the simulation kernel needs to evaluate each *characteristic expression* during every iteration in the simulation cycle and as many times as needed, the elaboration of a simultaneous statement must aid in performing such evaluations in a simple and efficient manner. For the purposes of explaining how this mechanism is used in Sierra, we use the semantics of linear forms of expressions as explained in [18] and shown in Figure 8.
Using the above mentioned semantics for transforming an expression into a function, every expression appearing in every simultaneous statement can be transformed into an equivalent function. We call this mechanism as *expression transformation.* The function can then be executed during simulation to determine the values of the unknowns in the continuous system. In VHDL-AMS the simultaneous statements can have arbitrary expressions on both the left hand side and the right hand side of the statement. But the analog simulation kernel requires that every unknown has a separate expression denoting its value at any point during simulation. This condition is enforced in Sierra by restricting the simultaneous statements to be of the form:

$$y = f(x_1, x_2, \cdots, x_n), \tag{3}$$

The equivalent function Ef of an expression E with respect to the name N of a value bearing object that appears in E is defined as follows:

```
function Ef(N:<typeofN>) return <typeofE> is
begin
    return E;
end Ef;
```

where Ef is a name unique to the expression and the name, <typeofN> is the type of the value denoted by the name N, and <typeofE> is the type of expression E.

Figure 8: Equivalent function of an expression E [18]

where y is the dependent variable(quantity) and $x_1, x_2, \cdots, x_n$ are the independent variables. Hence, the elaboration of the simultaneous statements imposes this restriction on the simultaneous statements that can appear in the text of the model. In Sierra, simple simultaneous statements can have arbitrary expression on the right hand side with the left hand side being a dependent unknown quantity or its derivative. The above mentioned mechanism is also applicable to ***simultaneous-if*** and ***simultaneous-case*** statements.

3.1.4 Break statements

Break statements are used in VHDL-AMS model to process discontinuities. Analog kernel is informed of the occurrence of discontinuity by break statements and proper handling is performed by finding new DC operating point before finding any further transient solution. The syntax of the ***concurrent break statement*** is:

[label :] **break** [break-list][sensitivity-clause][**when** condition];

Concurrent break statement represents a process containing a break statement whose syntax is:

[label :] **break** [break-list][**when** condition];

Here, break-list contains quantities and their new values that must be assigned to them as soon as discontinuity occurs. Occurrence of discontinuity is determined by the condition present in the break statement. In the absence of the condition, break statement is used for initialization of quantities. During the elaboration of break statement a boolean signal corresponding to a break statement is created and that signal is put on to the sensitivity list of the analog process. Objects of break-set and break-element are instantiated and quantities, expressions giving their new values and conditional expressions are stored in these objects. A break-element stores the quantity and expression giving new value for it in a break statement. A break-set contains linear list of break-elements. A linear list of break-sets is maintained to store all the break statements. During simulation, when the process containing a particular break set is executed and condition corresponding to that particular break-set becomes true then discontinuity is reported to the analog kernel by activating the break signal, which is already on the sensitivity list of the analog kernel. A new DC operating point is found by assigning the newly evaluated expressions for quantities stored in the break-elements of active break-sets.

3.1.5 Attributes of quantities

In VHDL-AMS, quantities represent values of the unknowns in a continuous system description. They can have several attributes as defined in [18]. Attributes that can be handled in Sierra have been presented here.

The *'above* attribute The *'above* attribute for a scalar quantity is represented as *Q'above(expr)*, where *Q* is a quantity and *expr* is an expression of the same type as that of *Q*. This attribute represents a boolean signal whose value is TRUE if the quantity *Q* is sufficiently above the value of the expression *expr*.
The analog solver is the driver of the implicit signal represented by the attribute *Q'above(expr)*. Using the *expression transformation* mechanism, the expression *expr* in the attribute is transformed into a function. The analog simulation kernel then evaluates this expression after determining every analog solution point and accordingly places the transactions on the drivers of the boolean signal. Since every textual appearance of this attribute represents a signal, an unique signal has to be formed during elaboration and the fanout for this signal is determined.

The *'dot* attribute Another very important attribute of a quantity is the *Q'dot* attributes that represents an implicit quantity. The value of this implicit quantity is the time differential of the prefix quantity *Q* at the time the attribute is evaluated. Elaboration of this attribute creates these implicit quantities in the system. Implicit quantities are also considered as unknowns in the continuous design. The simulation kernel in Sierra recognizes and solves for these differential quantities as though they were differential equations given by the model [10]. Hence elaboration of these differential quantities needs to identify the various differential quantities and transform them into differential equations. But these equations are implicit in the sense that they are not created by the text of the model.

4 The Analog Kernel

The core algorithm for analog simulation in Sierra is similar to that for SPICE adapted to the unique needs of VHDL-AMS. We describe how the class of simultaneous statements is handled in the simulator along with the requirements that need to be supported for the VHDL-AMS language.

4.1 Automatic Matrix Formulation

In this section, the ODAEs will be classified into different classes and procedure for dynamically generating the appropriate templates as described in [10]. A simultaneous statement expresses a quantity in terms of other unknowns. Its of the form

$$y = f(x) \tag{4}$$

These refined ODAEs that model the behavior of the components in a system by relating the quantities in the system, are known as branch constitutive equations (BCEs).

4.1.1 Generic elements for the BCEs

In a BCE $y = f(x)$, the quantity y can be an across, through or free quantity. The classification of the equations into generic elements is based on the type of quantity y. If y is a through quantity, then the equation is classified as a through equation. It is discretized and linearized producing the linear equivalent equation (from the BCE $I = f(x)$):

$$I_{n+1} - \sum_{j=1}^{k} \alpha_j^n x_j^{n+1} = I_s^n \tag{5}$$

where $I_s^n = I_n - \sum_{j=1}^{k} \alpha_j^n x_j^n$ and k is the size of vector x. In a similar fashion, we can arrive at the discretized and linear equivalent of an across equation(type of quantity y is across). Suppose y is a free quantity, we have a free equation and consider such an equation of the form

$$q_f = f(q_{fk}, v_l, i_m) \tag{6}$$

where $k, l, m \geq 0$ and q_{fk}, v_l, i_m are the different free quantities, across quantities and through quantities respectively in the free equation. It's discretized and linear equivalent is:

$$q_f^{n+1} - \sum_{j=1}^{k} \alpha_{fj}^{n}.q_{fj}^{n+1} - \sum_{j=1}^{l} \alpha_{vj}^{n}.v_j^{n+1} - \sum_{j=1}^{m} \alpha_{ij}^{n}.i_j^{n+1} = q_f s^n \tag{7}$$

where $q_{f_s}^n = q_f^n - \sum_{j=1}^{k} \alpha_{fj}^n q_{fj}^n - \sum_{j=1}^{l} \alpha_{vj}^n v_j^n - \sum_{j=1}^{m} \alpha_{ij}^n i_j^n$

4.1.2 Matrix Load Phase

A terminal is little more than the set of branch quantities declared using the name of that terminal in a branch quantity declaration. Each such branch quantity belongs to one other terminal as well; the one on the "other end" of the branch. Electrical nodes (or their analogy in other physical systems) are created when terminals are connected with other terminals in a structural description (an unconnected terminal is also a node). The laws of conservation at these terminals form the implicit characteristic expressions in the model.
The matrix loading is accomplished in two phases. During the first phase, the branch constitutive equations in the model are loaded into the matrix by traversing all the equations, During the run time elaboration [48], a terminal contribution data structure is built up for the purpose of storing information regarding the conservative equations in the system. This data structure is associated with every terminal storing the information regarding the through quantities incident on it, along with their orientation. A traversal of this data structure is sufficient to load the conservative equations in the system into the matrix.

4.2 Requirements of a VHDL-AMS analog kernel

Simultaneous statements are used in VHDL-AMS to express the equations that represent the behavior of a lumped parameter continuous system. The automatic matrix formulation stage described in section 4.1 is used to build the matrix at each time point. A list of features that need to be supported by the simulation kernel to implement the modified matrix formulation phase is as follows.

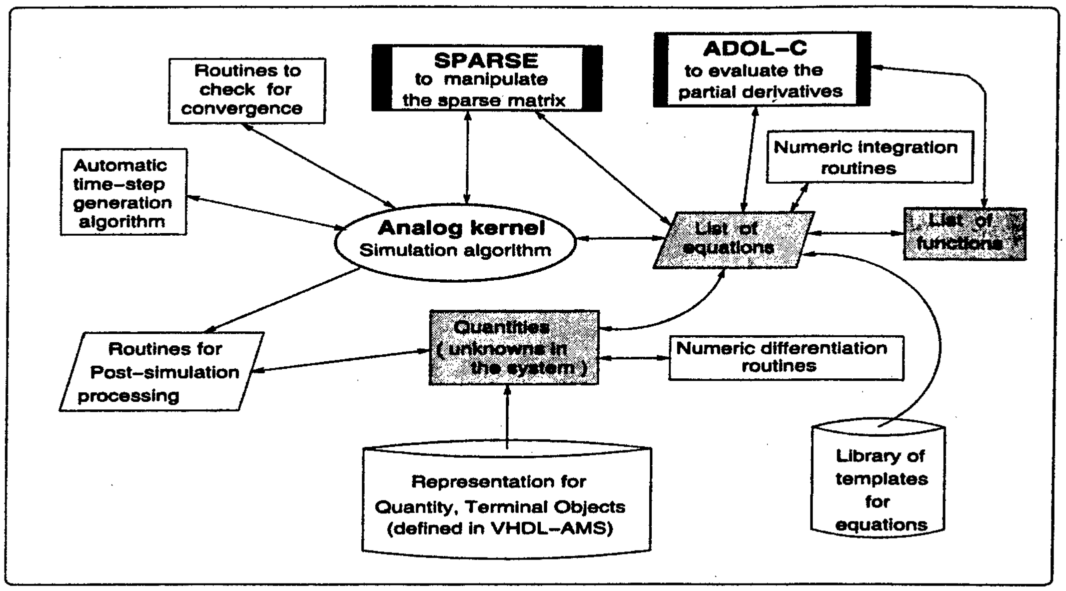
1. Generate the Conservation Laws implicitly
2. Define routines to perform numeric integration
3. Generate partial derivatives to construct the Jacobian
4. Support conditional simultaneous statements
5. Process Discontinuities occurring during simulation

4.3 Analog Kernel in Sierra

In this section, the analog simulation kernel in Sierra will be described. Figure 9 shows the environment with all the different packages which form the core of the kernel. The requirements in the previous section are supported in the following ways.

Generate the Conservation Laws implicitly: Mathematically, simultaneous statements represent a system of ODAEs that have to be solved for the unknowns at every time instant. Section 4.1 introduced the terminal contribution data structure which stores information regarding the through quantities incident on any terminal. A traversal of this data structure would give us all the conservative equations at every terminal in the system.

Define routines to perform numerical integration: In VHDL-AMS *Q'dot* represents the derivative of any quantity *Q* with respect to time. In order to perform the numerical integration(to convert a differential equation into an algebraic equation every simultaneous statement that has an ODAE is first transformed into an equivalent set of simultaneous statements using the principle: Every quantity in a simultaneous statement of the form *Q'dot* is replaced by a new quantity IQ(implicit quantity) in that statement and a new simultaneous statement of the form $IQ == Q'dot$ is generated implicitly.



SPARSE to manipulate the sparse matrix
ADOL-C to evaluate the partial derivatives
Routines to check for convergence
Automatic time-step generation algorithm
Numeric integration routines
Analog kernel Simulation algorithm
List of equations
List of functions
Routines for Post-simulation processing
Quantities (unknowns in the system)
Numeric differentiation routines
Representation for Quantity, Terminal Objects (defined in VHDL-AMS)
Library of templates for equations

Figure 9: Analog Simulation Environment

For example, consider a simultaneous statement that models the behavior of a linear inductor.

$$v == L * i'dot; \tag{8}$$

where v is the voltage across and i is the current through the inductor L. Applying principle 1, the transformed set of simultaneous statements for equation 8 are

$$\begin{aligned} v &== L * IQ; \qquad (9) \\ IQ &== i'dot; \qquad (10) \end{aligned}$$

Because of this transformation the differential equation that the kernel needs to support will always be of the form $IQ = Q'dot$. The conversion of this ODAE into an equivalent algebraic equation can be performed by the trapezoidal numerical integration method. For every equation of the form $IQ = Q'dot$ we can generate the template as follows. Consider a differential equation of the form $x' = f(x,t)$, where x' refers to the derivative of x with respect to time. The trapezoidal rule for this equation is given by

$$x_{n+1} = x_n + \frac{h}{2}(x'_{n+1} + x'_n) \tag{11}$$

where $(n+1)$ refers to the current iteration, n refers to the previous iteration and h refers to the current time-step. Now, the differential equation that the kernel should support is given by

$$IQ = Q'dot \tag{12}$$

Applying the trapezoidal rule to the above equation, we get

$$Q_{n+1} = Q_n + \frac{h}{2}(Q'_{n+1} + Q'_n) \tag{13}$$

Substituting equation 12 in equation 13, we get

Support the conditional simultaneous statements:

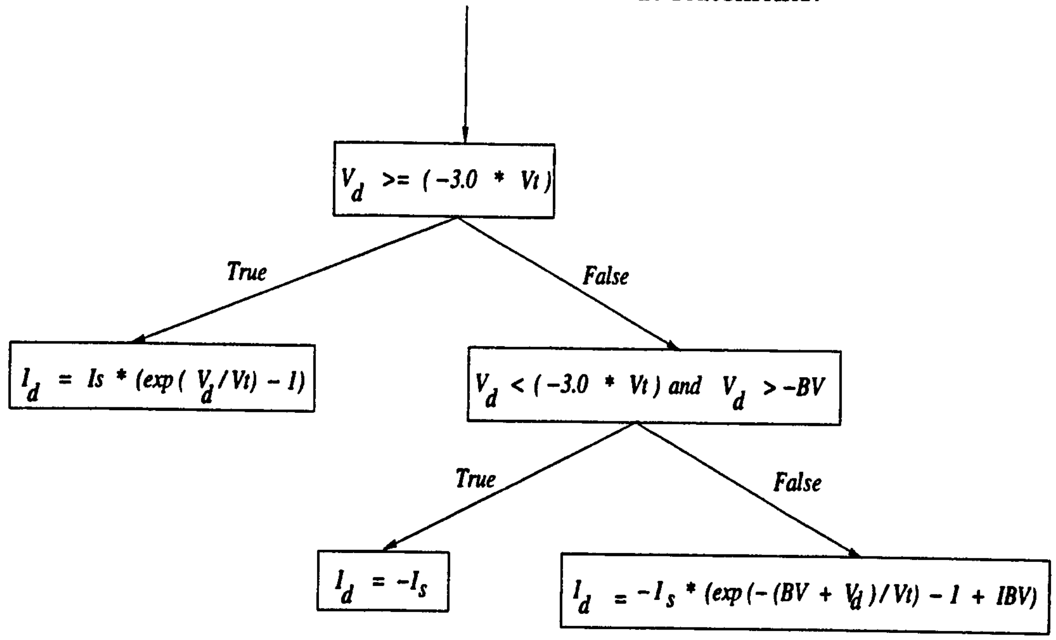

$V_d >= (-3.0 * Vt)$
True
False
$I_d = Is * (exp(V_d/Vt) - 1)$
$V_d < (-3.0 * Vt)$ and $V_d > -BV$
True
False
$I_d = -I_s$
$I_d = -I_s * (exp(-(BV + V_d)/Vt) - 1 + IBV)$

Figure 10: Binary tree structure for the simultaneous-if statement

$$Q_{n+1} = Q_n + \frac{h}{2}(IQ_{n+1} + IQ_n) \tag{14}$$

Rewriting the above equations, with all the unknowns on the left hand side of the equation and all the known values on the right hand side of the equation, we have

$$\frac{2}{h} Q_{n+1} - IQ_{n+1} = IQ_n + \frac{2}{h} Q_n \tag{15}$$

The above equations can be represented in the form $\alpha x = \beta$ as

$$\left[\frac{2}{h} \quad -1 \right] \begin{bmatrix} Q_{n+1} \\ IQ_{n+1} \end{bmatrix} = \left[IQ_n + \frac{2}{h} Q_n \right] \tag{16}$$

The analog kernel uses the above equation as the template for every differential equation in the system(which is always implicitly generated according to principle 1).

Generate the partial derivative to construct the Jacobian: The basic form of a template for a BCE of the form $y = f(x)$ is rewritten as

$$\left[1 \quad -\alpha_j^n \right] \begin{bmatrix} y^{n+1} \\ x_j^{n+1} \end{bmatrix} = \left[y_s^n \right] \tag{17}$$

In the above equation, α_j^n is a partial derivative of y with respect to the variable x_j at $x_j = x_j^n$. The matrix of all such partial derivatives(from the set of BCEs to be solved for) is called the Jacobian matrix. The analog kernel uses an automatic differentiation package called **ADOL-C(Automatic Differentiation by OverLoading in C++)** [43] to compute all the partial derivatives at every time instant. The partial derivatives, thus computed, are then loaded into the matrix using the templates defined for the different types of equations.

The simultaneous if statements and simultaneous case statements are the two types of conditional statements in VHDL-AMS. In our simulator, the simultaneous case statements are converted into equivalent simultaneous if statements and so the kernel needs to support only simultaneous if statements. At every iteration, the condition in these statements are evaluated first, and then, the appropriate statements have to be included in the set of ODAEs that need to be solved. This is implemented in the analog kernel using a binary tree structure. For example let us consider a pn-junction diode represented in VHDL-AMS:

```
If  ($V_d$ >= (−3.0 ∗ $Vt$))  use
    $I_d$ == $Is$ ∗ ($exp(V_d/Vt)$ − 1) ;
  elsif  (($V_d$ < (−3.0 ∗ $Vt$))  and  ($V_d$ > −$BV$))  use
    $I_d$ == −$Is$ ;
  else
    $I_d$ == −$Is$ ∗ ($exp(-(BV + V_d)/Vt)$ − 1 + $IBV$) ;
  enduse;
enduse;
```

where V_d(voltage across the diode) and I_d(current through the diode) are branch quantities, Vt is thermal voltage, Is is saturation current, BV is forward breakdown voltage and IBV is reverse breakdown voltage of the diode. The binary tree structure created by the simulator for this statement is shown in Figure 10.

Discontinuity Processing during simulation: In VHDL-AMS, $Q'above(E)$ describes a boolean signal which is assigned a value *true* if the value of the expression $Q - E$, is sufficiently greater than zero, while it is assigned *false* when value of $Q - E$, is sufficiently less than zero. Here E refers to any legal expression in VHDL-AMS. This is the mechanism by which analog to digital conversions are handled in VHDL-AMS. During the analog simulation, we need to check if any of the expressions $Q_i - E_i$ specified in the model becomes contradictory and take necessary action.

All the expressions E_i, are evaluated at the end of every integration step during the simulation. If the value of $Q_i - E_i$, lies on either side of zero before and after the integration step, we can conclude that the threshold crossing occurred between the present and the previous step. For rapidly changing quantities, the point of actual threshold cross might be faraway from the present time point. Hence, we perform a linear interpolation to locate the root of the discontinuity.

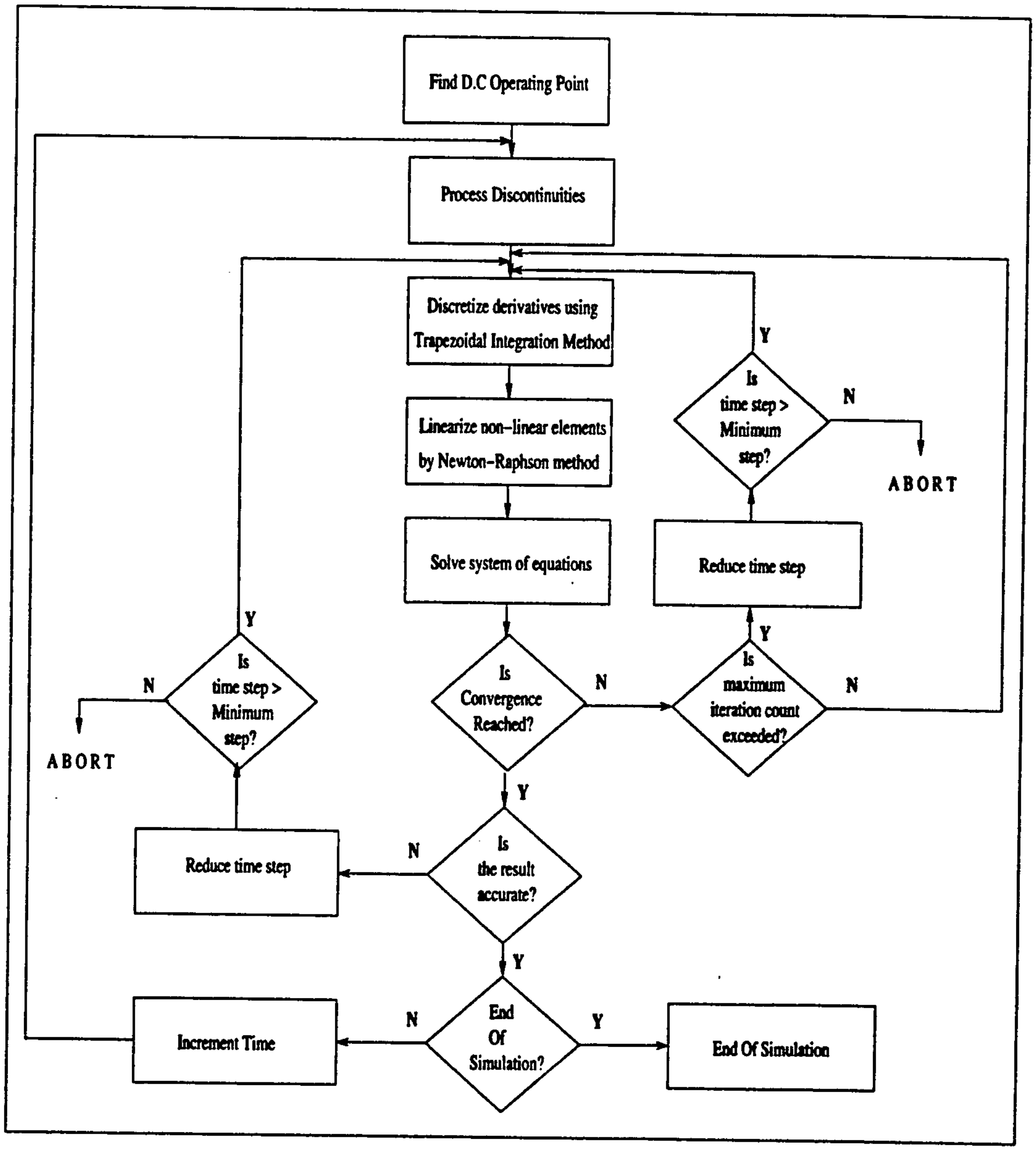
Suppose, the threshold cross occurred for any of the *'above* expressions in the model, the analog simulation is stopped, the digital processes sensitive to this *'above* signal are run, and once all the events at the current time are processed we come back to the analog process execution.

4.4 Flow of the analog simulation

Figure 11 shows a broad overview of the flow of analog simulation in Sierra. At the start of simulation, at time zero, when all the processes in the system are run once, a DC analysis of the analog system is performed to determine the initial values of the unknowns to be used in transient simulation. Then, we enter into the cycle of repeated execution of all the processes depending upon events posted on them with time advancing to the earliest time of an event. This constitutes the transient simulation of the analog process.

At every time point, the derivatives are discretized using the **trapezoidal integration** routine, the non-linearities are linearized by **Newton-Raphson** method(N-R), and the linear system of equations are solved using **L-U Decomposition** method of solution. Before the solution is accepted at any time point, we check for the convergence of the N-R method, and the truncation error introduced by the numerical integration algorithm. If convergence has not been reached in the required number of iterations, or if the truncation error in the solution is large, the solution at that time point is rejected and the time-step is reduced to find a new solution point. Otherwise, we proceed with the simulation.

The automatic time-step algorithm used in SPICE wherein, the time-step is reduced as indicated above, and the time-step is increased during stable regions in simulation where the convergence of the Newton-Raphson method is faster, and the error introduced by the numerical integration routine is lesser [44].



Find D.C Operating Point
Process Discontinuities
Discretize derivatives using Trapezoidal Integration Method
Linearize non-linear elements by Newton-Raphson method
Solve system of equations
Is Convergence Reached?
N
Y
Is maximum iteration count exceeded?
N
Y
Reduce time step
Is time step > Minimum step?
Y
N
ABORT
Is the result accurate?
N
Y
Reduce time step
Is time step > Minimum step?
Y
N
ABORT
End Of Simulation?
N
Y
Increment Time
End Of Simulation

Figure 11: Analog Simulation in Sierra

5 Improving Simulation Execution Speed

Analog simulation is a very time-consuming activity for even the fastest of computers. Even though the system of ordinary differential-algebraic equation (ODAE) sets that describe the pehavior of a lumped-parameter circuit results in sparse matrices, each point in the n-dimensional space (n being the number of quantities in the system) requires on the order of $n^3¿$ computations. Further, the data structures (i.e., matrix(s) and vectors) holding the values to be, processes must be renewed (or at least modified) for each timepoint in the solution. As a result, analog simulation proceeds at a much slower pace that discrete-event simulation. Combining the two forms (discrete and continuous time) in the same simulation results in the continuous time components of the model driving the overall simulation time.
We present two new approaches to reducing continuous-time simulations. One is to reduce the size (i.e., n) of the numerical problem being solved in the kernel for non-conserved systems of ODAE's. The other approach is incrementally build a new data structure at each timepoint, rather than flush the structures for the just-finished timepoint and build a new one for the next. We describe both of these approaches below.

5.1 An Equation Reduction Technique for VHDL-AMS Simulation

5.1.1 The Approach

Branch constitutive equations A conservative system in VHDL-AMS is modeled using branch constitutive equations. Branch constitutive equations are the equations needed for every branch in a conserved system. While modeling conserved system the modeler needs to provide branch constitutive equations for every branch in the system. Consider the Figure 12.

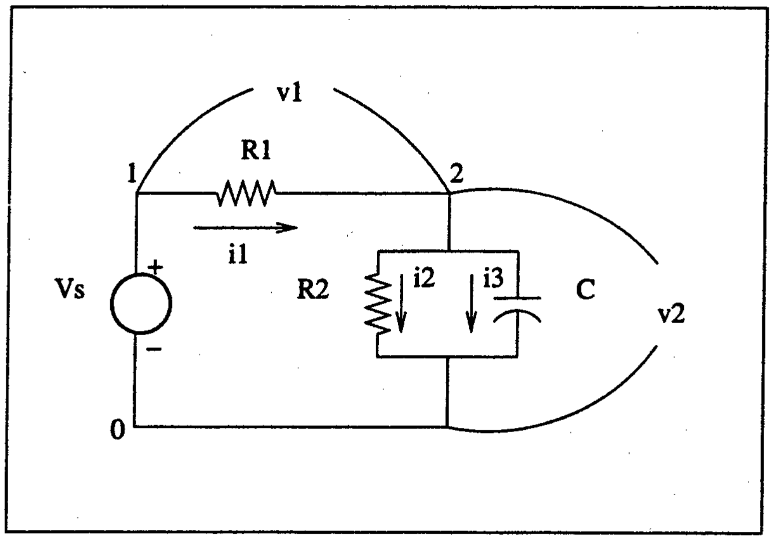


Figure 12: A simple electric network (conserved system) showing branch variables

In Figure 12 an electrical network is shown with four branches. This is a conserved system and the branch variables are the branch voltages and branch currents. Equivalent graph representation of this circuit is shown in Figure 13. In the graph representation each branch is associated with an equation representing the relationship between the branch variables for that particular branch. Such equation is known as branch constitutive equation.

Reduction technique A model needs to have branch constitutive equations for each branch in a conserved system. The number of branches in a circuit can become very high for large models and so can be the number of equations. Increased number of equations would cause slow simulation. The idea behind making simulation faster is that the modeler might not be interested in viewing the

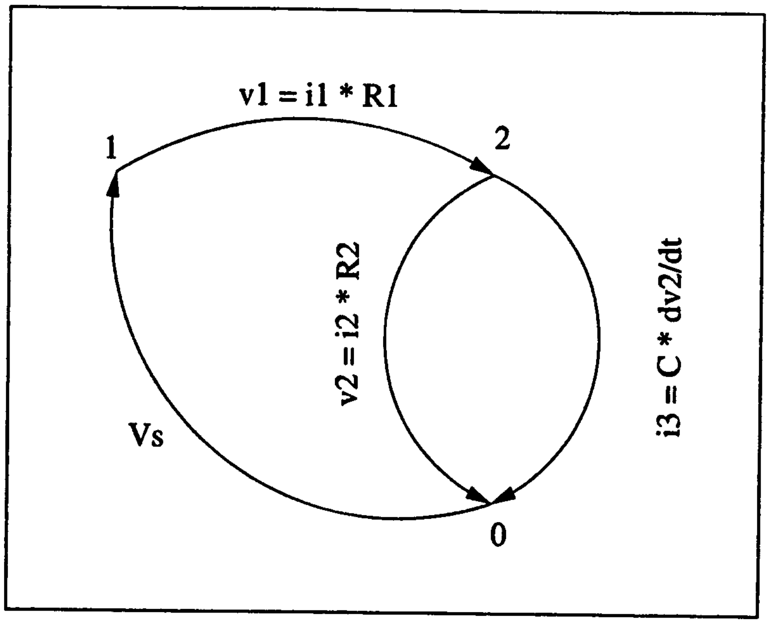


Figure 13: Equivalent graph representation of figure 12 showing branch constitutive equations

outputs of all the branch variables in the circuit. In that case using the principle of series and parallel collapsing of the branches we can obtain an equivalent circuit with reduced number of branches but different branch parameters values (like resistance, capacitance etc) now. By the term series and parallel collapsing we mean the replacement of a number of the same types of branches in series and parallel by an equivalent branch with different branch parameters. This task of branch reduction of a model is a one time activity and this reduces the number of branches and hence the number of equations which are needed. Since all needed equations are traversed, reduction of the number of equations makes this traversal faster and since this traversal is done for a number of times during simulation there is significant enhancement in speed by the reduction of the number of equations.

Graph representation To perform branch reduction the circuit is first translated into a graph $G(V, E)$, where V is the *vertex set* and E is the *edge set* of graph G. The sets V and E fully characterize the graph G. For a graph representing a conserved system, V contains the nodes of the system and E contains the weights and types of the edges connecting these nodes. An edge $e = (u, v)$ where $u \in V$ and $v \in V$, represents the connection between vertices u and v and the type and weight of this connection is associated with the edge e. Figure 13 represents an equivalent graph of an electrical circuit shown in Figure 12, which is a typical example of a conserved system.
A circuit needs to be transformed into an equivalent graph with each vertex denoting a node and each edge denoting the type and weight of the branch. The graph as implemented in this study is in the form of *adjacency list*. There is a list of vertices and each vertex has a list of edges that are connected to it. Parallel reduction is taken care of at the time of creation of the graph itself, i.e., at the time of inserting an edge in the graph if it is found that an edge of the same type, between the same nodes is already present in the graph then the edge being inserted is collapsed with the edge already present by parallel equivalence. Because of this there is only one edge of a kind between two vertices when the graph is finally ready for undergoing series collapsing. Once the graph is ready the algorithm shown in Figure 15 is used to perform series and parallel reduction of the branches. The input to the algorithm shown in Figure 15 is a graph of the circuit, which has been reduced by collapsing the parallel branches as much as possible. This input graph has no parallel branches at the start of the algorithm which can be collapsed. The algorithm starts by finding the nodes which can be omitted from the graph due to series collapsing. It then performs series reduction of the branches and because of that the newly created branch, which replaces the original ones can now be in parallel with another branch on which parallel reduction can be performed together with this newly created

branch. This can be clearly understood in Figure 14. In Figure 14 there is no possibility of parallel reduction for the input graph. But after series reduction when the intermediate node 2 is removed by collapsing the branches from node 2 then the newly obtained branch now comes in parallel with the branch between node 1 and 3 and hence parallel reduction can applied. We can see that at the start there were three edges of the same kind and removal of node 2 by series collapsing gave two branches between nodes 1 and 3 and then by parallel collapsing finally one branch with weight $R_{eq} = \frac{(R+R)*R}{(R+R)+R}$ is obtained. The algorithm shown in Figure 15 performs these reduction techniques and the steps are self explanatory.

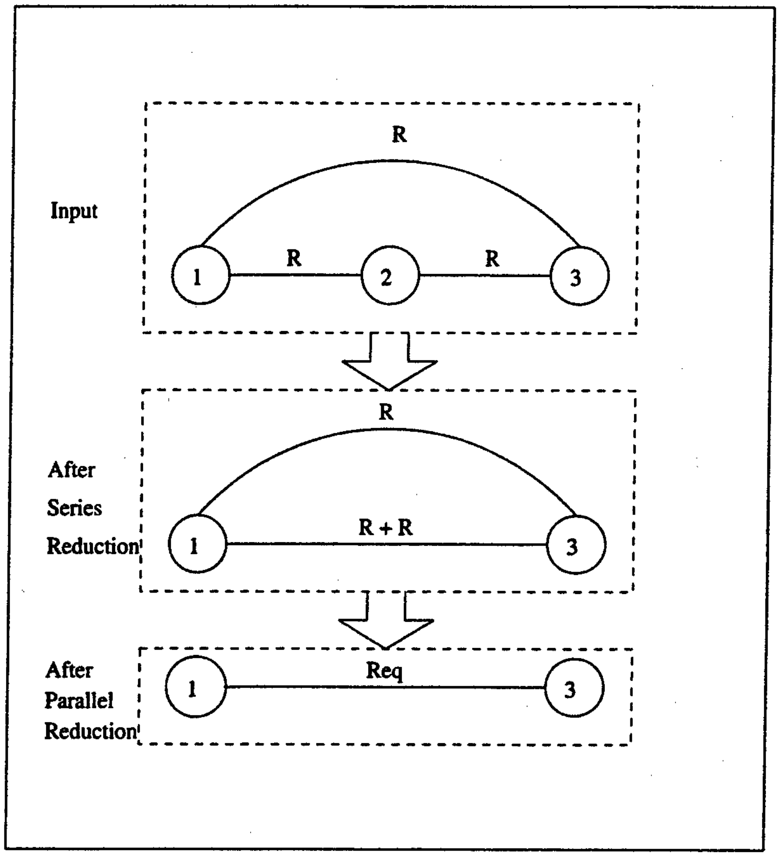


Figure 14: Series and parallel collapsing on input graph.

The complexity of this algorithm is O(size of *vertex-set*^{2} $*$ size of *edge-list* in vertex). Since *edge-list* in vertices are of the same order of the number of edges, the complexity of this algorithm for series and parallel reduction is $O(n^2 \ * \ m)$, where n is the number of vertices and m is the number of edges in the graph. Though the order is not linear, it is satisfactory to use this algorithm for branch reduction as it is a one time activity after elaboration in the mixed-signal simulation.

5.1.2 Reduction technique for non-conserved system

Free equations A non-conserved system is modeled using *free quantities.* Free quantities are value bearing objects in VHDL-AMS, which do not obey the laws of conservation. They are similar to any variables in a non-conserved system. The equations describing the behavior of these free quantities are termed *free equations.* The branch reduction technique described in the previous section is not applicable to non-conserved system as there is no concept of branches in such systems. For example let us consider a system of linear equations in x and y. And we just want to express some relationship between them. One such relationship could be as follows:

$$x \quad == \quad y \; ; \tag{18}$$

```
$v$ = start of the *vertex-list*
while ($v$ is not the end of *vertex-list*)
    if (the vertex $v$ has just two edges and they are of the same kind)
    then
        Create new edge from these two edges and insert in the *graph*;
        Remove all the edges in all the vertices of the *graph* having this vertex $v$;
        Remove the vertex $v$ from the *graph*;
        $v$ = head of the *vertex-list*;
    else
        $v$ = next vertex in the *vertex-list*;
    end if
end while
```

Figure 15: Algorithm for branch reduction in a conserved system

$$x \quad == \quad 2 + 4 * y\,; \tag{19}$$

The system represented by equations 18 and 19 depicts just a relationship between two variables. There is no notion of law of conservation here. Similarly, consider the system of equations that represents model of a bouncing ball. Let v represents the velocity of the ball and s represents the distance of the ball from the ground. Then the relationship between them is given by the following equations:

$$velocity : \quad v \quad == \quad s'dot\,; \tag{20}$$
$$acceleration : \; v'dot \quad == \quad -g\,; \tag{21}$$

where $s'dot \;=\; ds/dt$ and $v'dot \;=\; dv/dt$ and g is acceleration due to gravity. This system represented by equations 20 and 21 is a non-conserved system because the velocity of ball is reduced every time it hits the ground and finally it reaches the state of no motion. This handling of sudden change in the velocity is handled by *break* statement not shown here. This is a general description of a non-conserved system.

Reduction technique It is evident from the previous section that a non-conserved system is represented in altogether different way from a conserved system. Therefore the reduction techniques also differ in this case from that of the conserved system reduction technique. This section gives the reduction technique applicable on non-conserved systems.
Let us consider a non-conserved system of equations in three variables x, y and z. Let the behavior of the system be expressed by the following equations involving these variables:

$$x \;==\; F(y,\, z); \tag{22}$$
$$y \;==\; G(x,\, z); \tag{23}$$
$$z \;==\; H(x,\, y); \tag{24}$$

Substituting the expression of x from equation 22 into equations 23 and 24, we get the following set of equations for the system:

$$x \;==\; F(y,\, z); \tag{25}$$
$$y \;==\; G(F(y,\, z),\, z); \tag{26}$$
$$z \;==\; H(F(y,\, z),\, y); \tag{27}$$

Equations 26 and 27 are now in terms of two variables and if y and z are separable in these equations then finally the system can be expressed as:

$$x \;==\; F(y,\, z); \tag{28}$$

$$y == P(z); \quad (29)$$
$$z == Q(y); \quad (30)$$

where P and Q are new functions. Again by substituting y from equations 29 into equations 28 and 30 we get the system behavior as:

$$x == F(P(z),\ z); \quad (31)$$
$$y == P(z); \quad (32)$$
$$z == Q(P(z)); \quad (33)$$

As shown in the equations 31, 32 and 33, the behavior of the system can be obtained by just solving the equations 33 by numerical method and then symbolically substituting the value of z back into equations 31 and 32.
We can see that though the system is represented using three equations but after symbolic manipulation on the system of equations only one equation needs to be solved numerically and rest can be solved by substituting the value of the variables without any iteration.
Also, consider the following system of equations:

$$x == y; \quad (34)$$
$$y == z; \quad (35)$$
$$z == 5.0; \quad (36)$$

Variable substitution technique removes this kind of transitive relationship between the variables. In such transitive set of equations there is no need to put all the equations through general iterative method. Such kind of performance enhancement can be achieved by this technique very easily.
Therefore, to reduce the number of equations for a non-conserved system all free equations in the model being simulated is analyzed symbolically and by repeated variable substitution, if possible, the number of equations finally going to be the part of the numerical algorithm is reduced.

Representation and algorithm Symbolic analysis of equations needs a representation, which facilitates arithmetic operations on them symbolically. Representing equations in the form of tree is very suitable for this kind of analysis. In tree representation every node has two children except the leaf nodes. All variables and constants are the leaf nodes and all the operators are the intermediate nodes. Figure 16 shows a simple equation and its tree representation. This representation helps perform symbolic manipulation on equations. Once this kind of tree is obtained for every equation in the system the algorithm shown in Figure 17 is used to manipulate it symbolically and solve or reduce the system as applicable.

5.1.3 Equation Reduction Experimental Results

Evaluation Approach We ran a number of experiments both without the reduction method and with it. All simulation runs were performed on a Sun Blade-100 computer with SunOS Release 5.8 and the gcc version 2.95.3 compiler. All experimental data collected represents an average of five simulation runs each to ensure statistical integrity of results.
Because of the limitation faced by internal representation of characteristic expressions of VHDL-AMS models in Sierra tight integration of techniques to enhance the speed of the simulator was not possible. Hence to study these optimization techniques in conjunction with Sierra a different approach was taken. For branch reduction technique the environment which was used to compare the simulation results is given in Figure 19. In a similar way the environment for variable substitution technique is given in Figure 20.
Models needed for evaluating the performance enhancement technique were generated by scripts. Scope of this study was limited to linear circuits with non-dependent sources that are static in nature. So general circuits of R, C, L network with voltage source were created using script for study of branch reduction technique. For variable substitution, models with transitive and without transitive relationship were generated.

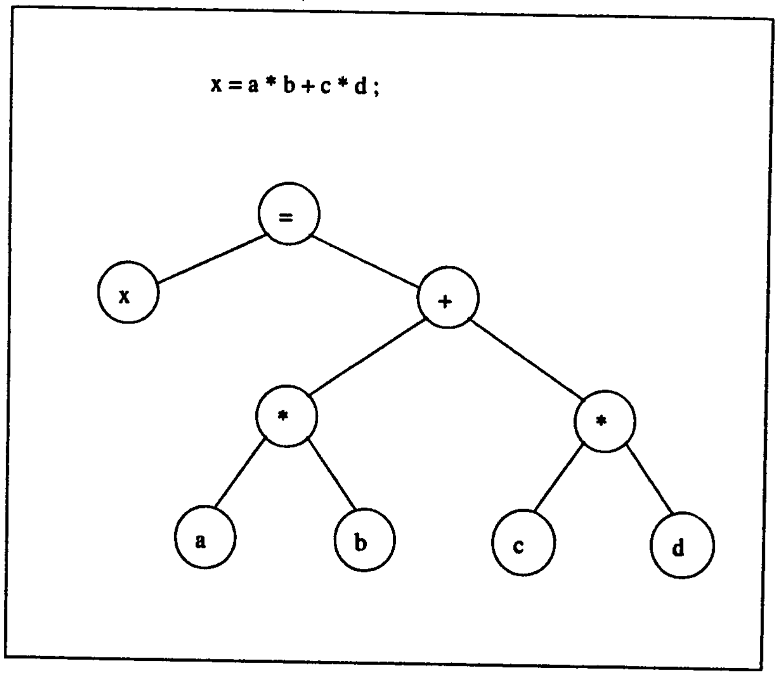


Figure 16: A simple equation and its representation in tree

For branch reduction algorithm the models needed were generated with varying number of branches of different elements, viz., resistor, capacitor, inductor and voltage source. Mixed signal models needing proper handling of A/D (Analog to Digital) and D/A (Digital to Analog) conversions were also generated by the model generator. Two such models are shown in figure 21 and 22. Figure 21 shows a bunch of resistors and capacitors interconnected with voltage source. This is pure analog model as there is no interaction with digital world. Figure 22 shows the interconnection between analog and digital parts of a simple circuit. The digital circuit is driven by the analog part of the circuit and hence need for proper handling of A/D conversion.
Models needed for equations reduction by variable substitution were also generated by scripts. For such models emphasis was mainly onto Ordinary Differential Algebraic equations as ODAEs cover big subset of systems which are generally encountered in real world. These models had randomly generated transitive relationships between quantities in the form of ODAEs
Data obtained by simulation run of various models is presented here. Two techniques have been studied in conjunction with Sierra to enhance mixed signal simulator. First, improvement obtained by branch reduction technique is given and then enhancement due to variable substitution technique is presented.

Results for the Branch Reduction Technique We first examine the performance of the reduction method on the scalable circuit. We ran successive simulation on circuits with a variable number of terminals but otherwise representing the circuit of the form given in Figure 21. In all cases the number of equations reduced to eight thereby providing a consistent view of how the reduction method performs as a circuit grows. As stated in previous chapter, reduction in branch equations should speed-up the mixed signal simulator if there is reduction of even one branch in the equation set obtained after running the branch collapsing algorithm over the set of branch equations . The evidence shown in Table 1 supports this assertion because speedup as given in the equation 37 is always greater than 1.0. In the Table1 reduction factor is defined by equation 38.

$$Speed\ up = \frac{simulation\ time\ without\ branch\ reduction}{simulation\ time\ with\ branch\ reduction} \quad (37)$$

```
Optimize
{
  for $\forall e \in EqnSet$
    Call $simplify(e)$
  end for
  for $\forall e \in EqnSet$
    Find occurrence of LHS variable of $e$ in $E$ - {e}
    and store them in $occurList$

    for $\forall h \in occurList$
      if ($h$ is the left child of its root) then
        if ($root(h)$ has more than one variable) then
          $rearrange(root(h))$
        else
          continue /* next iteration */
        end if
      end if
      newTree = copy of subtree rooted at RHS of $e$
      replace $h$ by newTree and adjust tree
      update variable numbers in the root of newTree
      call $simplify(root(newTree))$
    end for
  end for
}
```

Figure 17: Algorithm for free equation optimization

```
Simplify(equation tree $e$)
{
  find occurrence of LHS of $e$ in the RHS of $e$
  and store them in hitList
  for $\forall h \in$ hitList
    move up the tree from $h$ until a $+$ or $-$ is encountered
    and at each intermediate node get the coefficient of LHS of $e$
    if (separable equation) then
      modify tree of $e$
      update the coefficient
    end if
    keep accumulating the coefficient for each $h$
  end for
  if (coefficient is $-1$) then
    there is no need of LHS of $e$ and this is put into the
    list removedList and a new variable from RHS of $e$ is
    put at the LHS if possible.
  end if
  coefficient for LHS of $e$ is updated
  if (only one variable in $e$) then
    $e$ has been solved for LHS of $e$ so put it in the solvedList
    and remove from the removedList if present
  end if
}
```

Figure 18: Algorithm for simplifying one equation

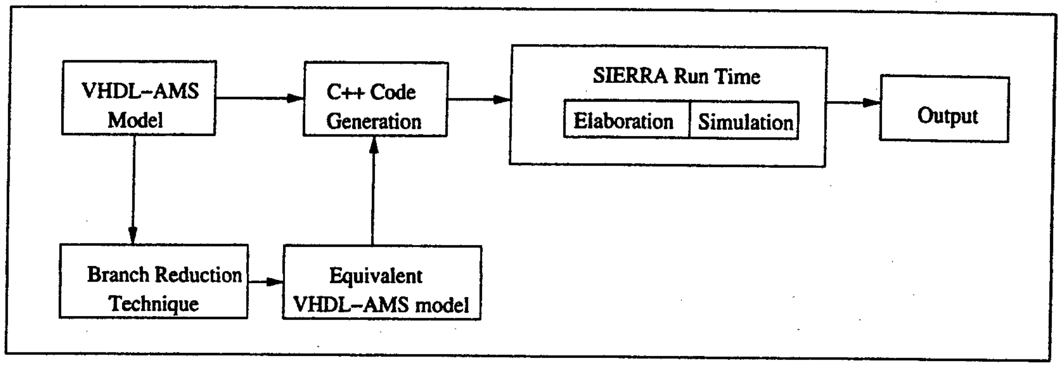


Figure 19: Methodology for studying branch reduction technique together with Sierra, a mixed signal simulator

Model	No. of Branch Equations	Simulation Time Without Reduction (sec)	Simulation Time With Reduction (sec)	Reduction Factor
A1	12	1.23	0.99	1.5
B1	23	1.87	1.02	2.9
C1	45	3.39	1.03	5.6
D1	89	7.03	1.04	11.1
E1	177	18.29	1.03	22.1
G1	353	37.61	1.05	44.1
H1	705	90.58	1.07	88.1

Table 1: Performance Improvement Because of Branch Reduction Technique

$$Reduction\ Factor = \frac{No.\ of\ original\ branch\ equations}{No.\ of\ reduced\ branch\ equations} \tag{38}$$

Figure 23 shows the simulation time with increasing number of branches in conserved system. The best fit of the polynomial for this curve found using statistical analysis is of $O(N^3)$. This is what was expected. Figure 24 represents curves with reduction technique and without reduction technique. Here since after reduction every time the number of branches obtained was fixed, the curve with reduction technique is almost constant. This curve is presented here to point out the tremendous performance enhancement achieved if lots of parallel branches can be reduced. The two methods used for obtaining these curves were statistically compared for finding the confidence level with which they can be said to differ from each other. The mean difference of simulation time for these two methods are statistically analyzed to find the confidence interval and confidence level. To claim that the curves are different the confidence interval for the mean difference should not include zero. The maximum confidence level which does not include zero in the confidence interval of the mean difference of simulation time has been shown in the Table 2. Corresponding graph has been shown in Figure 25. Figure 26 shows the amount of time taken by the branch reduction technique to achieve a particular degree of speedup. Of course, the degree of speedup achieved is dictated by the system. This curve just depicts the amount of time taken by the reduction algorithm for a particular speedup factor if permissible by the system description.

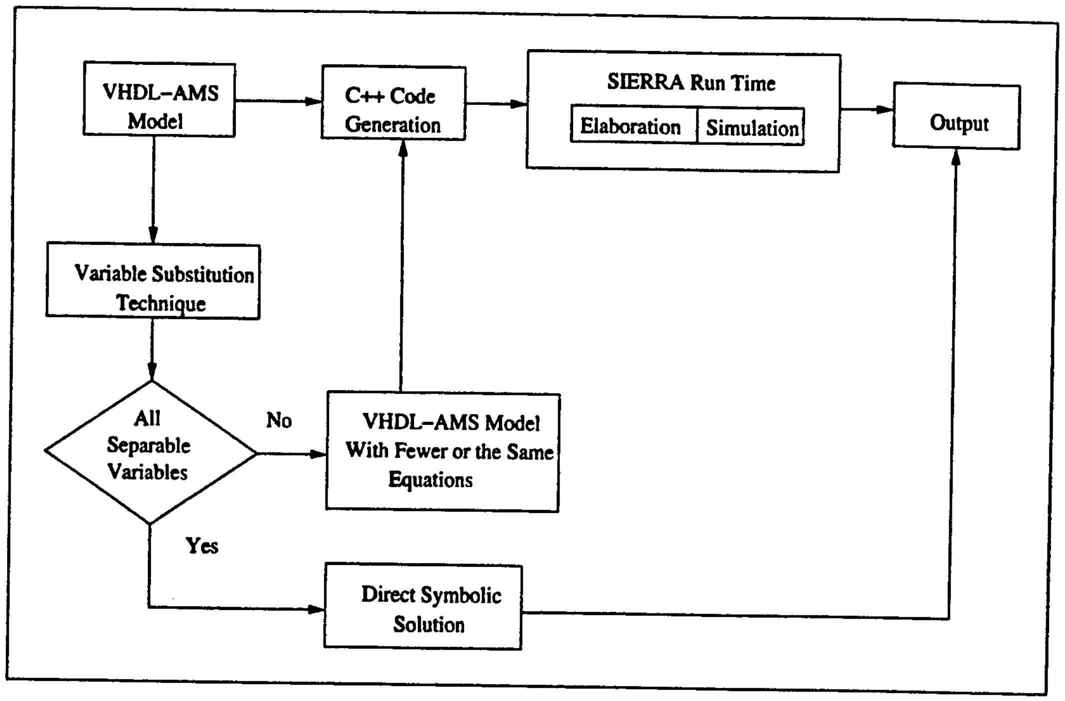


Figure 20: Methodology for studying variable substitution technique together with Sierra, a mixed signal simulator

Results for Variable Substitution Technique The variable substitution technique should improve the performance of the mixed signal simulator depending upon the degree of linearity of the mixed signal model that is being simulated. Data shown in Table 3 supports this assertion. As expected, the method of symbolic solution of a system of equations, improves the simulator's performance depending upon its degree of linearity. It is evident from the plot of number of equations vs. speed-up factor that for very large number of equations in the model the speedup will not be very significant and that is because of dominance of I/O over actual useful section of substitution technique.

Figure 27 represents simulation time with respect to the number of free equations in the model without using any optimization technique. The best fit polynomial for this curve found using statistical analysis has $O(N^3)$, where N is the number of equations. Figure 28 shows the reduction in simulation time obtained using the equation reduction technique. Again, the mean difference of simulation time for these two methods is statistically analyzed to find the confidence interval and confidence level. To claim that the curves are different the confidence interval for the mean difference should not include zero. The maximum confidence level which does not include zero in the confidence interval of the mean difference of simulation time shown in the Table 4. Corresponding graph has been shown in Figure 29. Performance enhancement using equation reduction technique is obvious from the curve. Figure 30 shows the speedup factor with increasing number of equations. As stated above with increasing number of equations the speedup factor decreases because of the I/O overhead. Results obtained are in conformation with our assertion that reduction in equations would lead to speedup of the mixed signal simulation.

Model	No. of Branch Equations	Mean difference of simulation time (sec)	Confidence Interval for the Mean	Confidence Level %
A1	12	0.24	(0.05, 0.43)	90
B1	23	0.85	(0.68, 1.01)	99
C1	45	2.33	(2.00, 2.67)	99
D1	89	5.99	(5.25, 6.73)	99
E1	177	17.27	(16.40, 18.13)	99
G1	353	36.56	(34.74, 38.38)	99
H1	705	89.51	(85.09, 93.93)	99

Table 2: Confidence interval for mean difference of simulation time with branch reduction technique and without that for data from Table 1 Technique

Model	No. Equations	Simulation Time With Iterative Method (sec)	Simulation Time With Symbolic Method (sec)	Speedup Factor
A2	5	0.61	0.02	32.78
B2	10	0.82	0.04	23.75
C2	20	1.41	0.05	27.01
D2	40	2.27	0.12	18.62
E2	80	4.44	0.40	11.17
F2	160	9.51	1.53	6.20
G2	320	22.50	6.40	3.52

Table 3: Performance Improvement Because of Symbolic Solution Technique

Model	No. Equations	Mean difference of Simulation Time (sec)	Confidence Interval for the Mean	Confidence Level %
A2	5	0.59	(0.53, 0.65)	99
B2	10	0.78	(0.69, 0.87)	99
C2	20	1.36	(1.32, 1.40)	99
D2	40	2.16	(2.10, 2.21)	99
E2	80	4.03	(3.95, 4.11)	99
F2	160	8.10	(7.59, 8.62)	99
G2	320	16.10	(15.36, 16.84)	99

Table 4: Confidence interval for mean difference of simulation time with reduction technique and without that from Table 3.

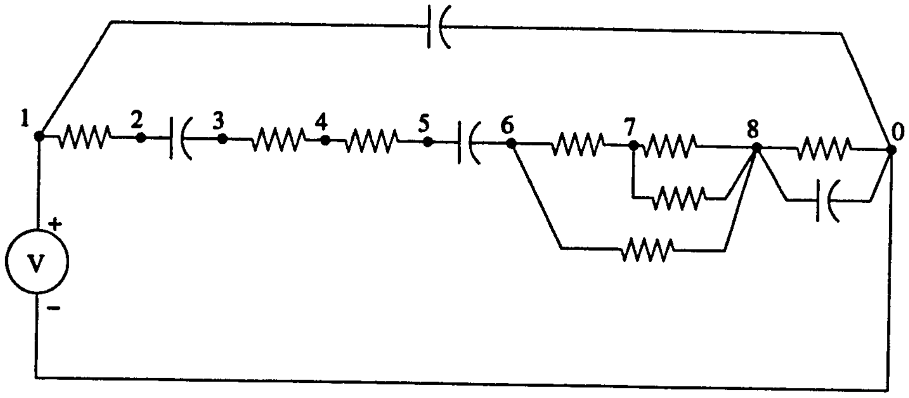


Figure 21: A simple analog model of resistor and capacitor network

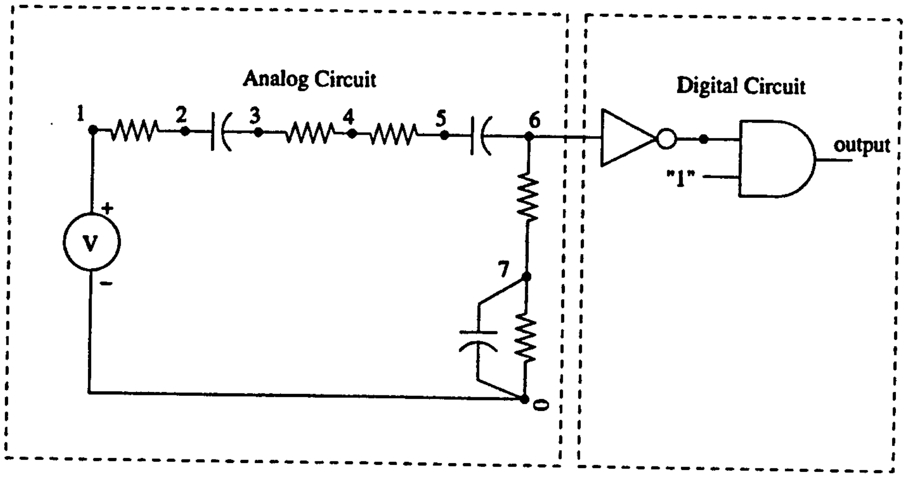


Figure 22: A simple mixed signal model with analog circuit driving digital gates

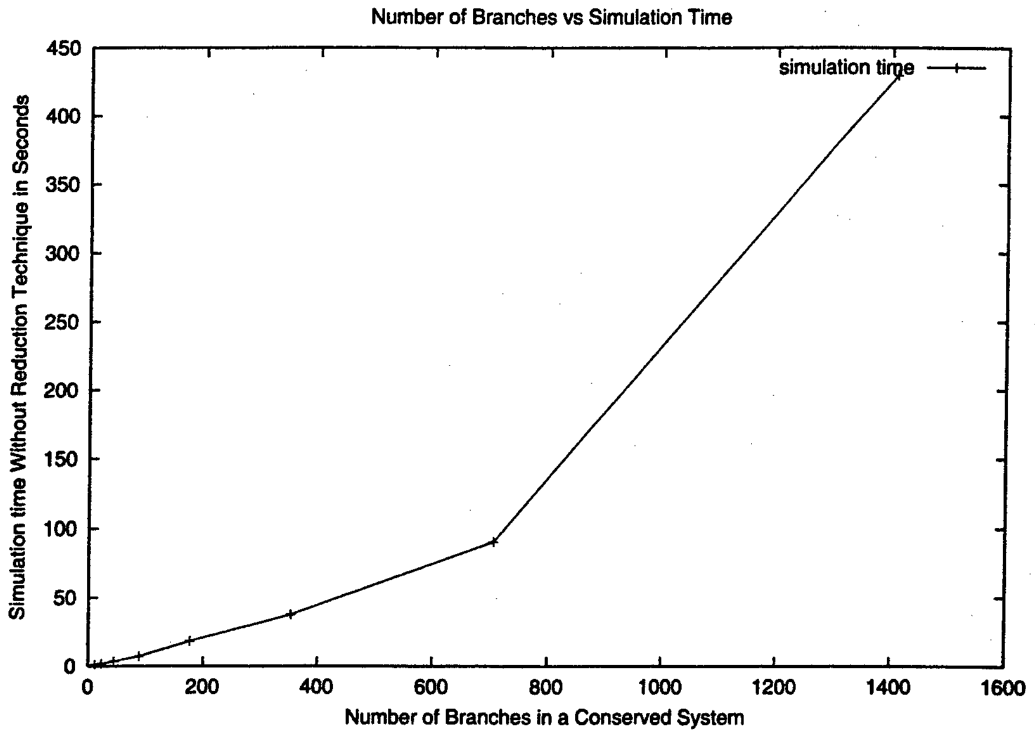


Figure 23: Increase in simulation time with number of branch-equations for scalable model

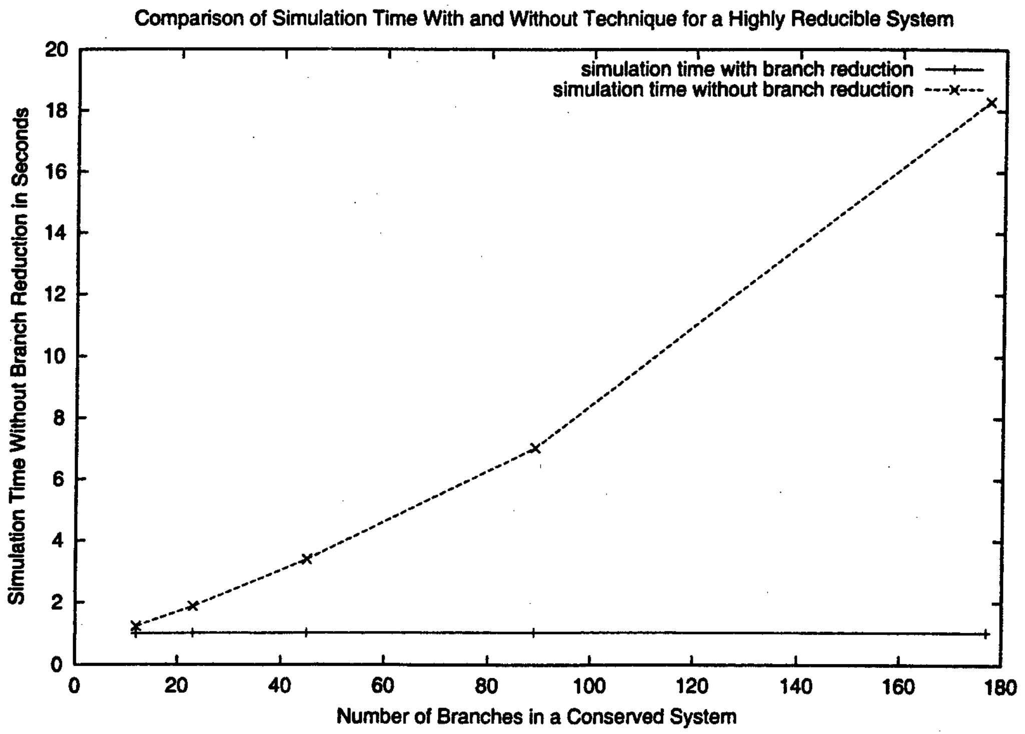


Figure 24: Comparison of simulation time with and without reduction technique

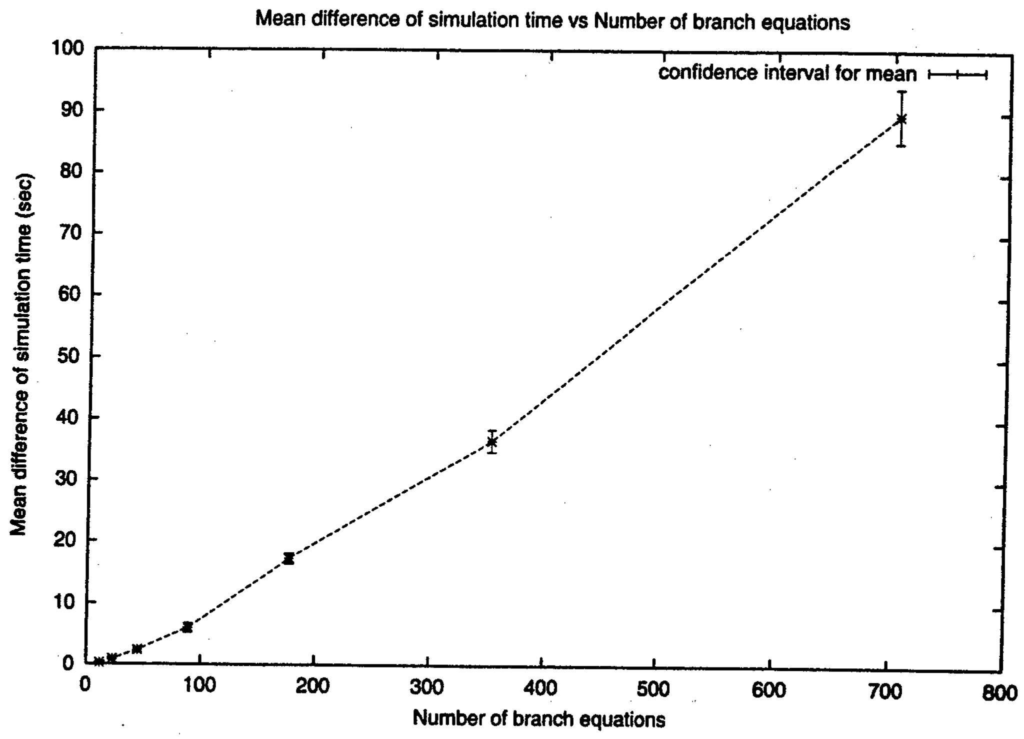


Figure 25: Graph showing confidence intervals for mean difference of simulation time for various branch equations.

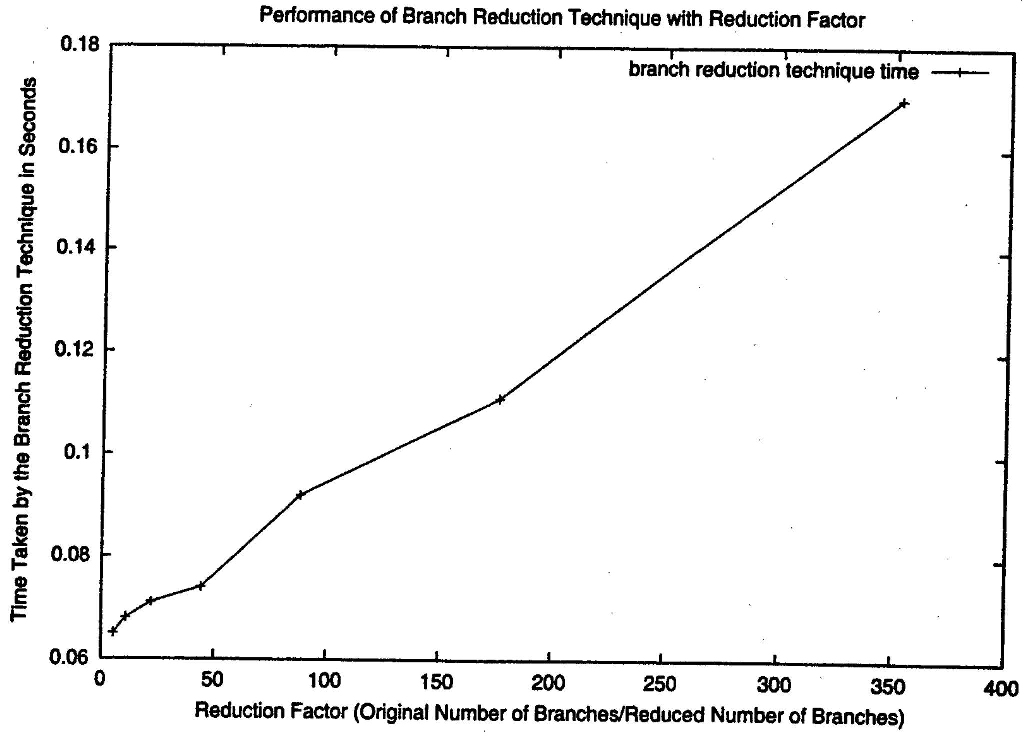


Figure 26: Study of branch reduction technique with branch reduction factor

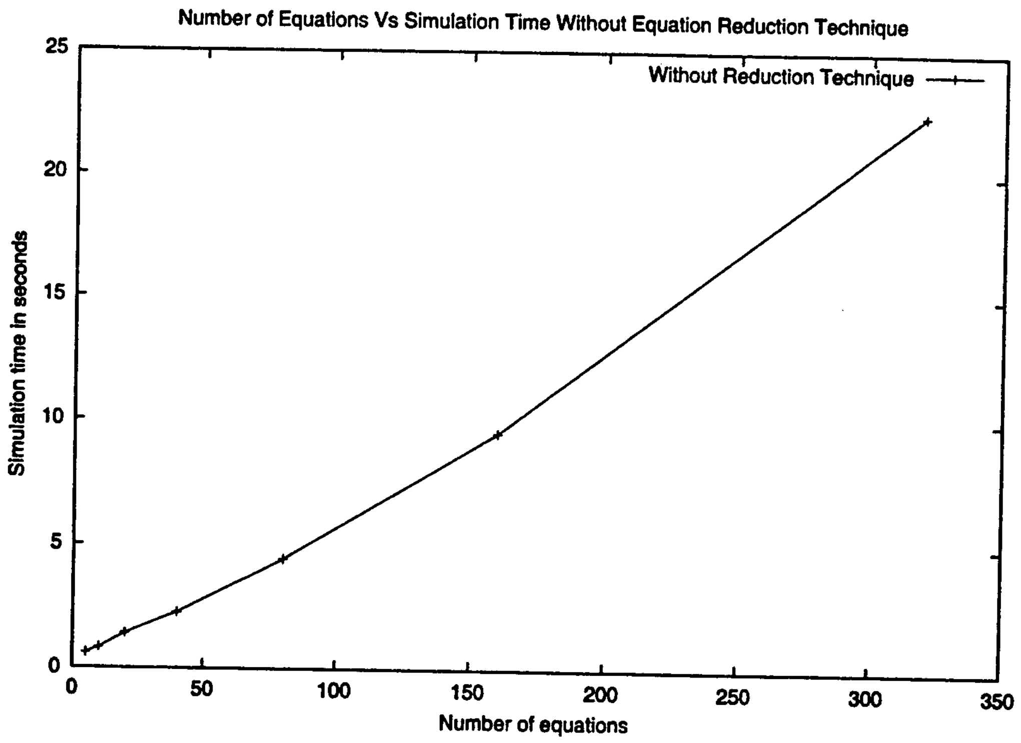


Figure 27: Increase in simulation time with number of equations

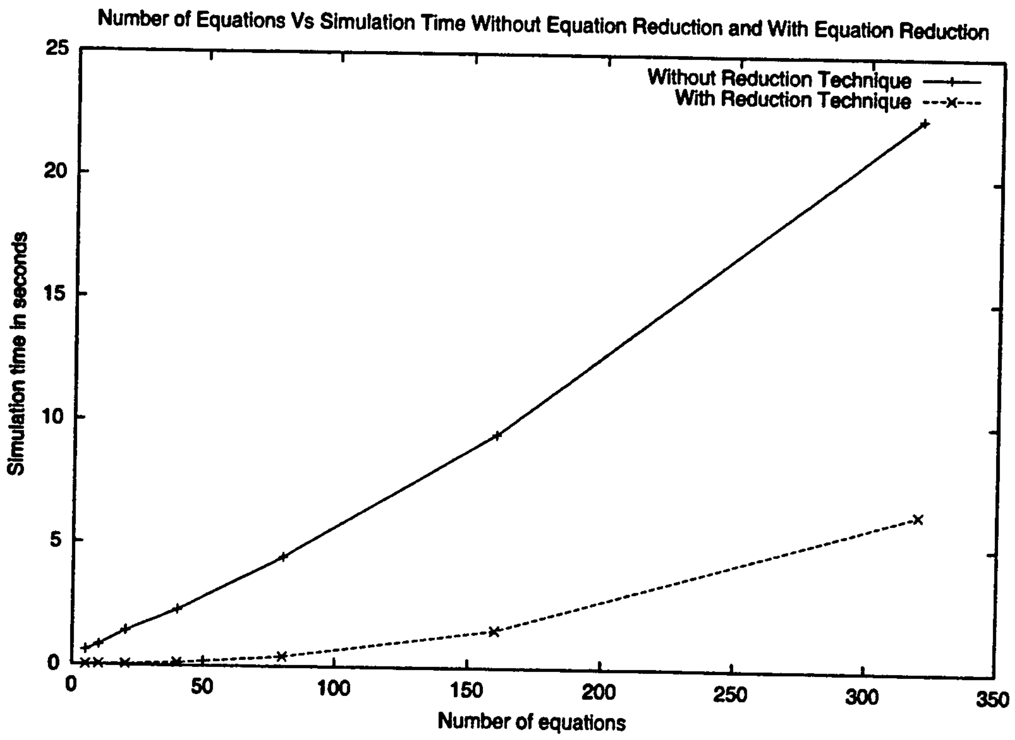


Figure 28: Comparison of simulation time with and without reduction technique

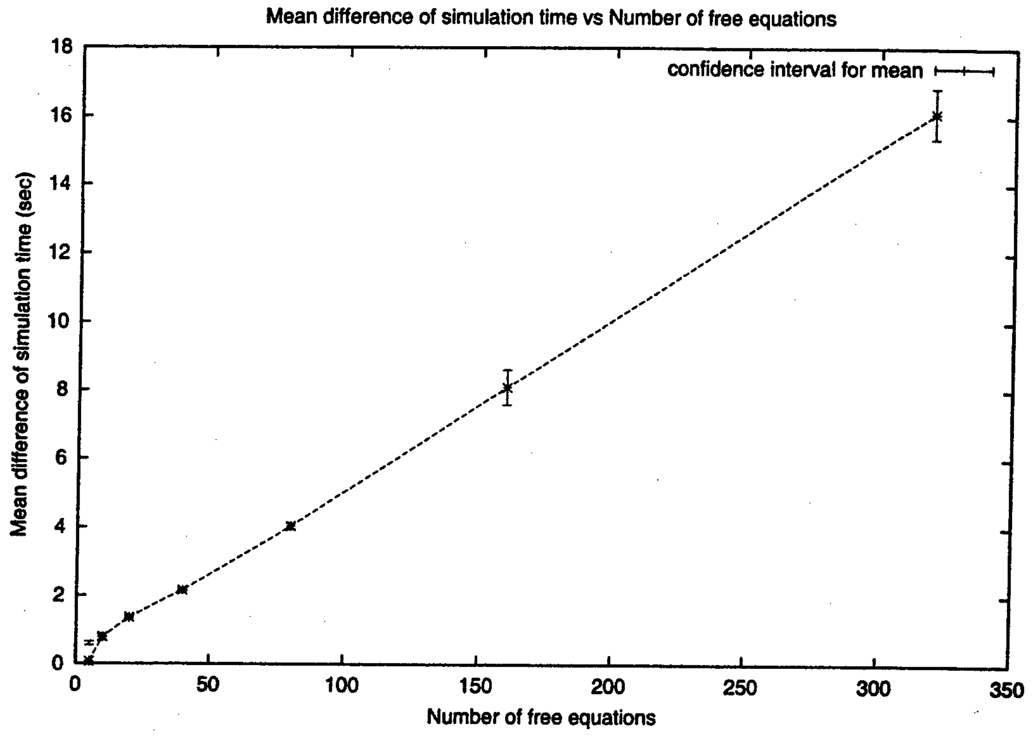


Figure 29: Graph showing confidence intervals for mean difference of simulation time with different number of free equations.

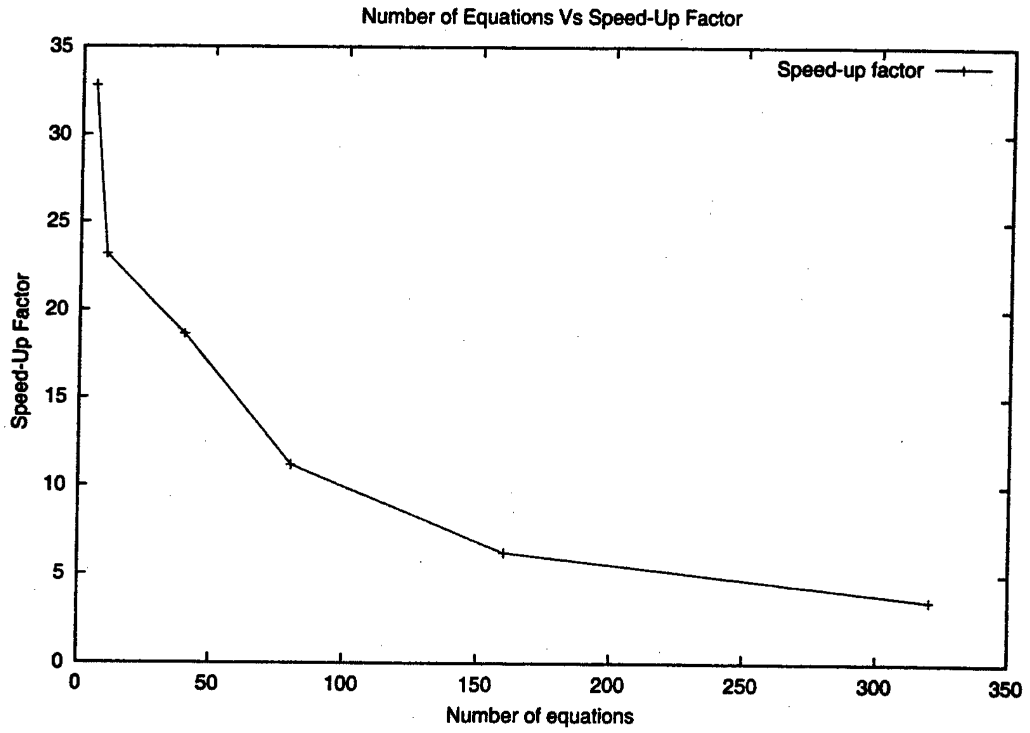


Figure 30: Speed-up factor variation with number of equations because of reduction technique

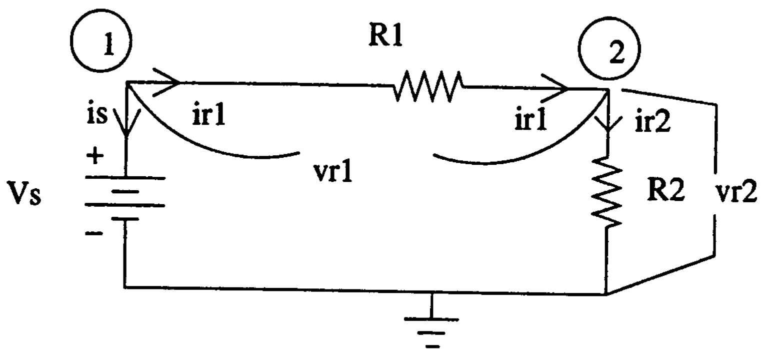


Figure 31: A voltage divider circuit to demonstrate the Sierra matrix build phase. R1 = 100 ohms, R2 = 200 ohms, Vs = 5V

5.2 Reducing the Matrix Build Time

5.2.1 Matrix Build in Sierra

In Sierra continuous-time simulation works by first loading all the elements into the matrix and then solving the matrix. Equations into three classes, namely, *across, through, differential and free*. This classification is based on the type of the left hand side quantity in the simultaneous equation representing the DAE. Then during the matrix build phase, appropriate load routine is called depending on the equation type. The stamps in SPICE automatically take into account the conservation (KCL) equations. The requirements for VHDL-AMS demand these conservative equations be created implicitly in the kernel. The modeler just needs to model the branch constitutive equations. The matrix build phase builds the matrix in two parts:

- Loading the *conservative equations* - All the conservative equations or KCLs in case of electrical circuits are loaded into the matrix.
- Loading the *branch constitutive equations* - All the branch constitutive equation corresponding to the simultaneous statements or the *explicit set* [18] are loaded into the matrix. Partial derivatives are calculated in accordance with Newton-Raphson method to load the proper values into the matrix corresponding to this equation (Building up the Jacobian matrix).

Sierra performs a matrix build at every time point or iteration during simulation as determined by the time step. At every time step during the simulation it starts from a null matrix and then builds the whole matrix by loading the conservative equations and the branch constitutive equations. However this might not be required as a few of the enteries don't change in the left hand side matrix or the A matrix. Thus part of the matrix A does not depend on the iteration. So, instead of starting with an empty matrix, we start with an initial matrix and then incrementally build the matrix. This is the basic idea behind CEO and MSM approaches mentioned below. This leads to a reduction in the matrix build time. The next sections describe each of the optimization approaches investigated in this thesis.

For example, consider the in Figure 31 and a VHDL-AMS model for it in Figure 32. For this model, the A matrix or the left hand side matrix corresponds to:

$$A = \begin{pmatrix} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \\ \mathbf{1} & ... & ... & 1 & ... & 1 \\ \mathbf{2} & ... & ... & -1 & 1 & ... \\ \mathbf{3} & 1 & -1 & -100 & ... & ... \\ \mathbf{4} & ... & 1 & ... & -200 & ... \\ \mathbf{5} & 1 & ... & ... & ... & ... \end{pmatrix}$$

The enteries in bold show the rows and column numbers.They are not part of the actual matrix. Rows 1 and 2 in the above matrix correspond to the KCL equations at node 1 and 2 as shown in Figure 31. Rows 3, 4 and 5 correspond to the branch equations corresponding to the three branches

```
PACKAGE electricalSystem IS
        NATURE electrical IS real ACROSS real THROUGH
        Ground reference;
        FUNCTION SIN(X : real) RETURN real;
END PACKAGE electricalSystem;
use work.electricalsystem.all;

–entity declaration

ENTITY voltage-divider IS
END voltage-divider;

–architecture declaration

ARCHITECTURE behavior OF voltage-divider IS
        terminal n1, n2 : electrical;
        quantity vs across is through n1;
        quantity vr1 across ir1 through n1 to n2;
        quantity vr2 across ir2 through n2;
        constant R1 : REAL := 100.0;
        constant R2 : REAL := 200.0;

        BEGIN
           BE1: vR1 == iR1 * R1;
           BE2: vR2 == iR2 * R2;
           vsrc : vs == 5.0
        END ARCHITECTURE behavior;
```

Figure 32: VHDL-AMS description of the circuit shown in Figure 31

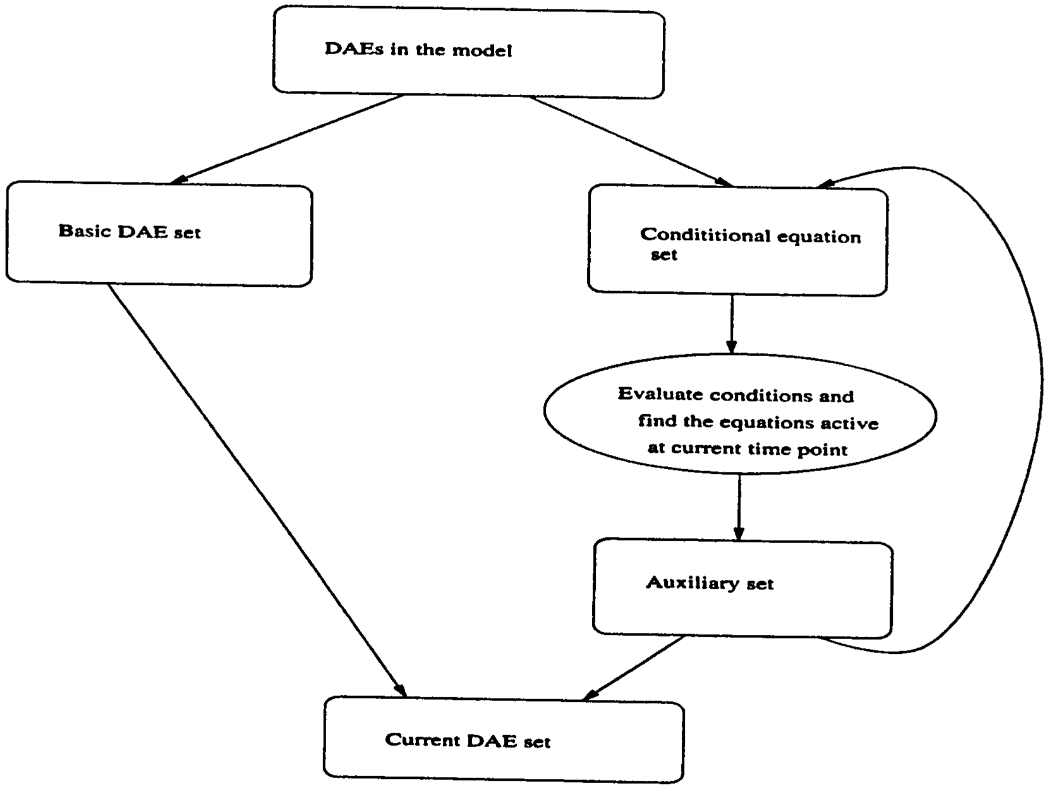

DAEs in the model
Basic DAE set
Condititional equation set
Evaluate conditions and find the equations active at current time point
Auxiliary set
Current DAE set

Figure 33: Classifying the equations into a base set and a auxiliary set.

shown in Figure 31. Row 3 is the branch equation for resistor R1, row 4 is the branch equation for resistor R2 and row 5 is the branch equation for the voltage source Vs.
The vector of unknown quantities (X) corresponds to the following vector:
$X = (\ v1 \quad v2 \quad ir1 \quad ir2 \quad is\ )^T$
Here v1 and v2 are the node voltages at nodes 1 and 2 in Figure 31.
The right hand side vector has a single non-zero entry corresponding to the voltage source:
$B = (\ 0 \quad 0 \quad 0 \quad 0 \quad 5\ )^T$
The matrix build phase creates the A and B matrix at every time step and solves for the unknowns, that is, matrix X. In the conventional approach, all enteries in A are reloaded at every time point irrespective of whether they change or not during the simulation. The approaches presented below are all targeted at improving the time to rebuild A at each time point.

Method 1 - Equation Set Optimization (ESO) After elaboration of the VHDL-AMS description, we have the set of equations describing the system. Some of these equations are part of the conditional set as governed by the ***simultaneous if*** statements in the input model. During simulation, the condition is evaluated and the appropriate equations are added to the equation set depending on the outcome of the condition evaluation. Thus at every time point we have to perform a process to gather all the equations. Also before loading these equations into the matrix, there is a set up phase. This set up phase essentially performs two tasks: 1) allocates branch or row where the equation will be entered into the matrix. 2) allocate pointers to positions in the matrix where the contribution to this equation will be entered. This is needed so that next time when we load the same equation, we know where to load the values. However this is done for every equation present in the system at current time point. The ESO technique attempts to speedup this set up phase of matrix build. The set up phase as mentioned above need not be done for all the equations in the system at every time point. The above process is modified by this approach by dividing the equations into two sets - a *Base Set* and a *Conditional Set*. Base set consists of the equations that will take part in simulation always. The equations that are not part of the ***simultaneous if***

statements constitute our base set. The equations inside the simultaneous if statements are part of the conditional set. After evaluation of the condition in the conditional set, we form the ***auxiliary set*** of equations depending on the outcome of the condition. This auxiliary set can change during simulation and thus is determined at every iteration. The current set at any time step is given by the union of the base set and the auxiliary set. This is shown in Figure 33. Having divided the set of equations into a base and auxiliary set, we can now speedup the setup phase. Now the two tasks in the setup phase needs to be performed only for the auxiliary set as its dynamic. However for the base set, we know that they will be a part of the current set throughout simulation, and hence we just need to allocate branch and allocate pointers for the equations in the base set only once. This might lead to a considerable speedup if the number of equations in the base set is considerable higher than the auxiliary set, which is the case typically. This forms the ESO technique. The ESO approach is represented by the procedure shown in Figure 34.
The results from the application of this approach have been reported in the next chapter and they show considerable performance improvement over the naive approach of having just a single equation set.

```
setup_Matrixbuild()

Input : Current DAE set which consists of a
    base set and a auxiliary set
Output: Performs initialization for loading the
    equations into the matrix later

  - This is done just once during the first iteration
  for all equations $\epsilon$ base set
      allocate_branch() - this determines the row for
          this equation in the matrix
      allocate_pointers() - this allocates positions in
          the matrix where the contribution due to this equation
          will be entered and returns pointers to those locations
  end for

  - This is done during every iteration
  for all equations $\epsilon$ auxiliary set
      allocate_branch()
      allocate_pointers()
  end for

end setup_Matrixbuild
```

Figure 34: Modified setup phase for equation set optimization (ESO) approach. In Sierra, setup was done for the entire equation set at every time point. Now setup will be just done once for the base set and will be done every time for the auxiliary set.

Method 2 - Conservative Equations Optimization (CEO) As explained in Section 4.1, matrix build phase consists of loading the conservative equations into the matrix. This corresponds to loading the KCL equations at node 1 and 2 in Figure 31. SPICE uses an element stamp method which automatically takes into account the KCL contributions due to the element. VHDL-AMS does not have a notion of an element or a component. Every element or a component is specified through a simultaneous statement. Thus a component like resistor can be specified either as :

$$v == i * r; \tag{39}$$

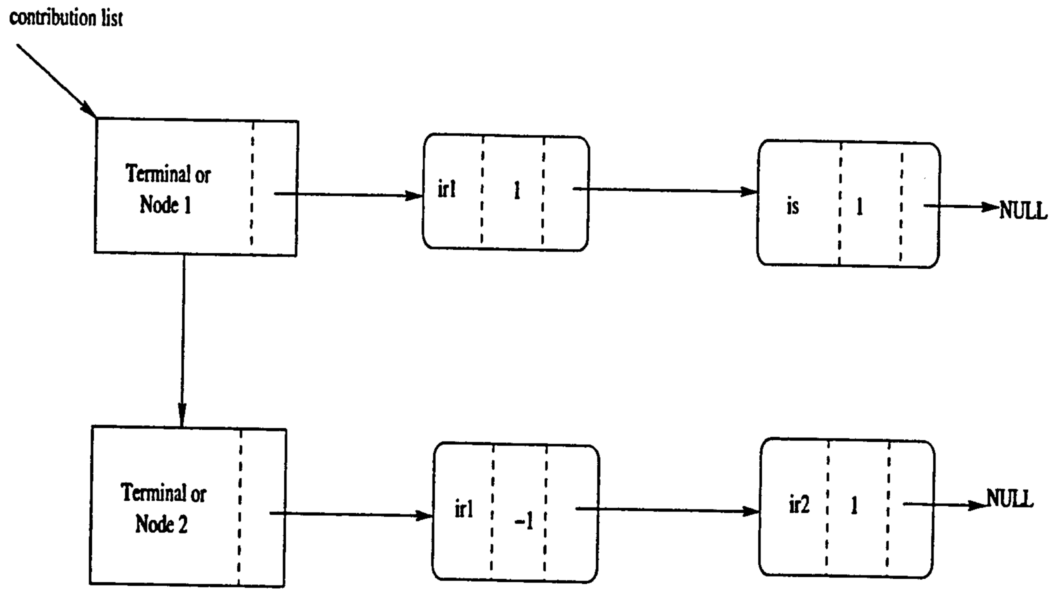


Figure 35: Structure of Contribution list created for loading KCL for circuit in Figure 31

or

$$i == v/r; \tag{40}$$

where v is the voltage across the resistor, i is the current through the resistor and r is the resistance. Thus unlike SPICE where from the first letter you can find the component type, its difficult to determine whether an equation represents a resistor in VHDL-AMS. The approach here has to be more general. Thus we classify the equations into *across, through or free equation* depending on the type of the quantity on the left hand side of the equation. The branch equations can be loaded but that still leaves us with implicitly generating the conservative equations.
This optimization approach tries to optimize this loading of KCL equations into the matrix. It was observed that the set of KCL equations do not change during simulation for most of the practical circuits. Thus instead of loading KCL equations at every time point as was done in Sierra, this approach just loads all the KCL equations once at the start of simulation. This approach worked very well as will be demonstrated by the results in the next chapter. As can be seen from matrix A shown earlier, quite a few enteries correspond to these KCL equations. Now we preserve these enteries for the next matrix build and we just have to incrementally load the branch conservative equations. Thus we have reduced the number of equations to be loaded into the matrix by the number of KCL equations. The savings obtained by this approach can be substantial as the number of nodes and the number of branches increases.
A terminal contribution list is created for loading the conservative equations. This list for the circuit in Figure 31 is shown in Figure 35. This list is used to enter the KCL equations into the matrix for nodes 1 and 2. The algorithm for doing conservative equation optimization is given in Figure 36. Thus during the start of simulation we load the KCL equations and bind the enteries corresponding to the KCL equations with the matrix. During the next iteration now, we initialize our matrix with these KCL enteries and then load the rest of the branch equations. Thus instead of starting with an empty matrix, we now start with a matrix which has already been partially created. Thus we achieve a significant speedup in matrix build time, which leads to reduction in the total simulation time.

This approach however needs to be used with a little caution. In SPICE language, this approach would have worked very well as the equation set cannot change dynamically during simulation. However VHDL-AMS allows the modeler to introduce or delete equations depending on a condition, through *simultaneous if* statements. So the KCL equations might also change during simulation if a new branch is added or an existing branch is completely removed from the circuit. Extending this approach to handle this case has been suggested for future work as it was observed that most of the practical circuits have the same conservative equations during simulation. Note here, that this approach would work fine if the equation of an existing element between two nodes is conditionally replaced by another equation during simulation.

```
incremental_matrixbuild()

Input : Current DAE set which consists of a
    base set and a auxiliary set and a contribution list
Output: Builds the matrix to be solved by
    analog solver for finding unknowns

 – Loading the conservative equations
 – This is done just once during the first iteration
 for all terminals ϵ contribution list
     add the contribution for each through quantity
         incident on the terminal
     associate the enteries with the matrix
         –this is done for initializing the matrix with
         –these enteries during subsequent iterations.
 end for

 – initialize matrix with KCL equations.
 initialize_matrix()

 – Loading the branch equations
 – This is done during every iteration
 for all equations ϵ current DAE set
     load the equation – loading this branch equation
 end for

end incremental_matrixbuild
```

Figure 36: Optimized matrix build phase. In Sierra, every iteration was started with an empty matrix and both the branch equations and KCL equations were loaded at every time point. With the ESO optimization, we load the conservative equations only once and then during succesive iterations we initialize our matrix with these KCL equations. So we start with an initial matrix and just have to load the branch equations.

Below we show the matrix with the KCL equations corresponding to circuit in Figure 31 loaded in the matrix. We start with this matrix now during successive iterations instead of the empty matrix. The branch equations are then added to this matrix. So for this simple circuit, the number of equations that need to be loaded in the matrix at every time point have been reduced to three as opposed to five in the naive approach.

$$A = \left(\begin{array}{cccccc} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \\ \mathbf{1} & ... & ... & 1 & ... & 1 \\ \mathbf{2} & ... & ... & -1 & 1 & ... \\ \mathbf{3} & ... & ... & ... & ... & ... \\ \mathbf{4} & ... & ... & ... & ... & ... \\ \mathbf{5} & ... & ... & ... & ... & ... \end{array}\right)$$

Next optimization approach builds upon the incremental matrix approach of this section and also adds the capability to utilize the best solution method depending on the characteristic of the equation set.

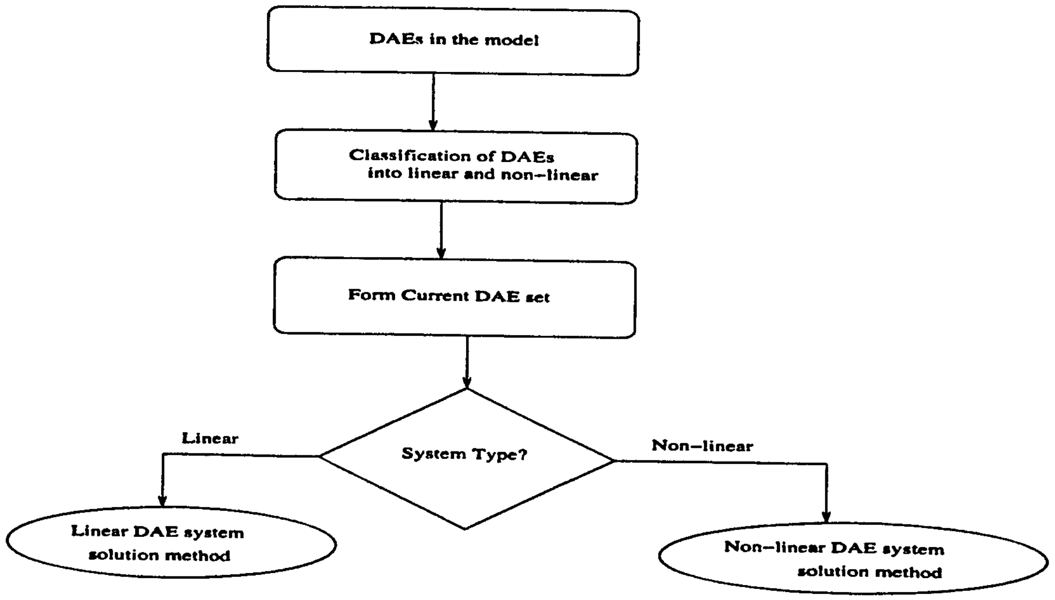

DAEs in the model
Classification of DAEs into linear and non-linear
Form Current DAE set
System Type?
Linear
Non-linear
Linear DAE system solution method
Non-linear DAE system solution method

Figure 37: DAE system solution process

Method 3 - Multiple Solution Methods (MSM) This method is an extension of an approach suggested by [34] and demonstrated in the SEAMS project. We take advantage of separating the linear part of the solution from the nonlinear part and apply the minimum-time solver to each. The process of simulation involves finding a consistent solution of the DAE set at all time points until the simulation ends. At any timepoint, the system is described by the base equation set and a auxiliary set. The auxiliary set is a result of the evaluation of the conditions. The current DAE set is formed by the union of base set and an auxiliary set that needs to be solved. The type of the equation set is determined to be linear and non-linear. If the system is linear, then we employ a linear solution process. If there are one or more nonlinear equations, then the system becomes nonlinear, and this requires a nonlinear system solution. The nonlinear system solution process is expensive as it requires the calculation of Jacobian matrix. Thus if we can determine that the equation set is linear, then there is no need to go through the expensive nonlinear solution process. The two separate solution paths are shown in Figure 37 and are described in the following sections. Also this approach is easily extensible to add new solution methods. Various algorithms can be used and tried out either interactively through the user or by the simulation algorithm for achieving a more solution or for doing a faster solution.

b. Linear system solution

Figure 38 shows the steps in solving a linear system of equations. The linear system of DAEs is first integrated to obtain a linear algebraic set of equations. Either a direct method or an iterative method can be used for solving this linear system of equations. In SPICE, Sierra or other existing continuous time simulators most often use the direct method. But, it has been observed that certain iterative methods [35] perform better on certain large sparse linear sytems. VHDL-AMS allowing modeling in different domains and its possible that matrices generated from other systems might have certain properties that favor iterative methods. The availability of multiple solution methods provides the flexibility for the user to choose a method that is more appropriate to his descriptions. A direct method is used to solve the equations in a fixed and finite number of steps. An iterative method on the other hand starts with an initial guess of the solution and iteratively refines the solution until convergence. For a linear system, the left hand side matrix or the A matrix is time invariant. The classification into a linear system obviates the need to form the Jacobian matrix, which has considerable computation overhead. If the solution does not converge, the timestep is reduced to one-eight of the current step. Tolerances are used to control the accuracy and the iterations performed.

c. Non-linear system solution

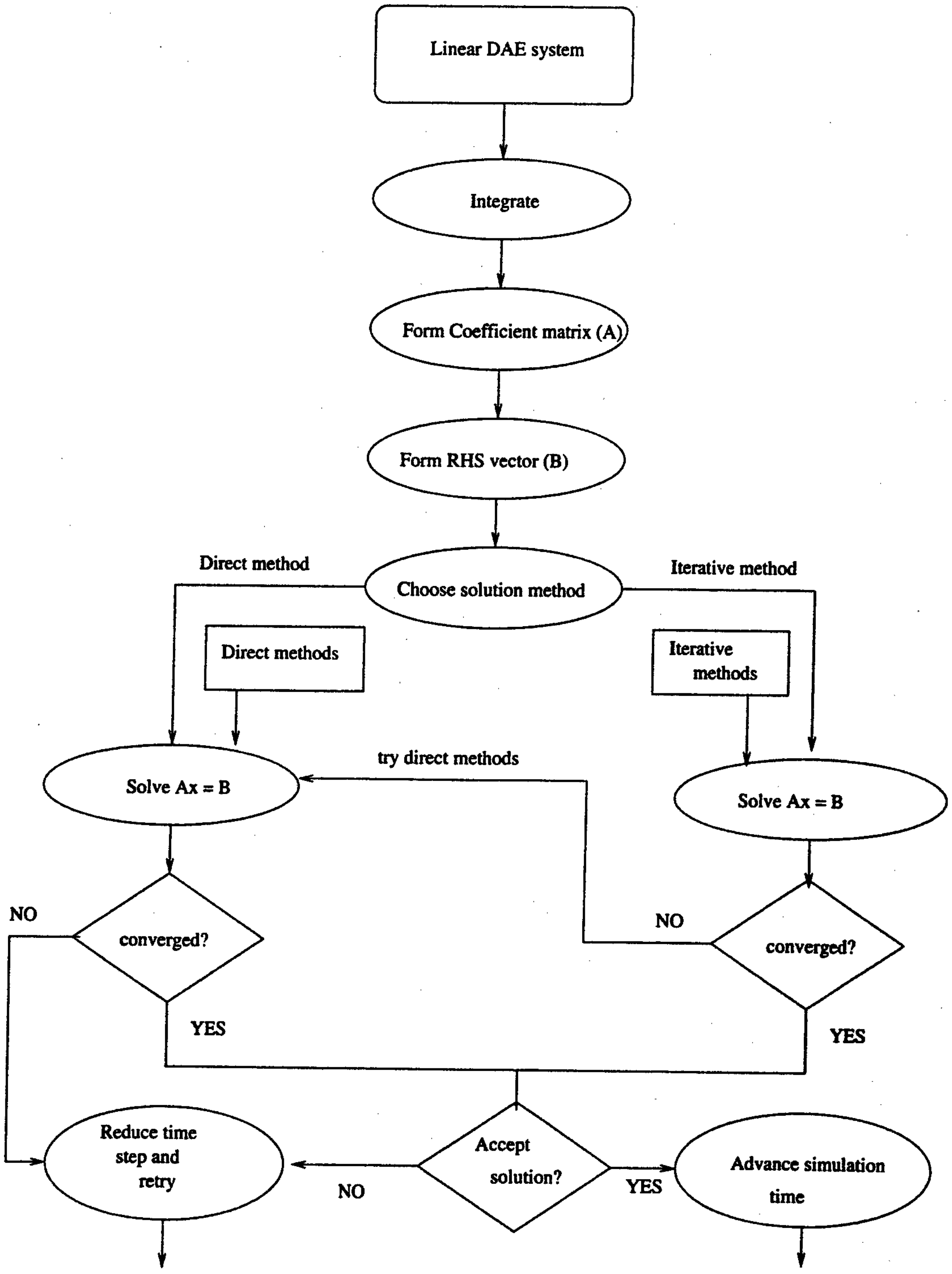


Figure 38: Linear system solution method [34]

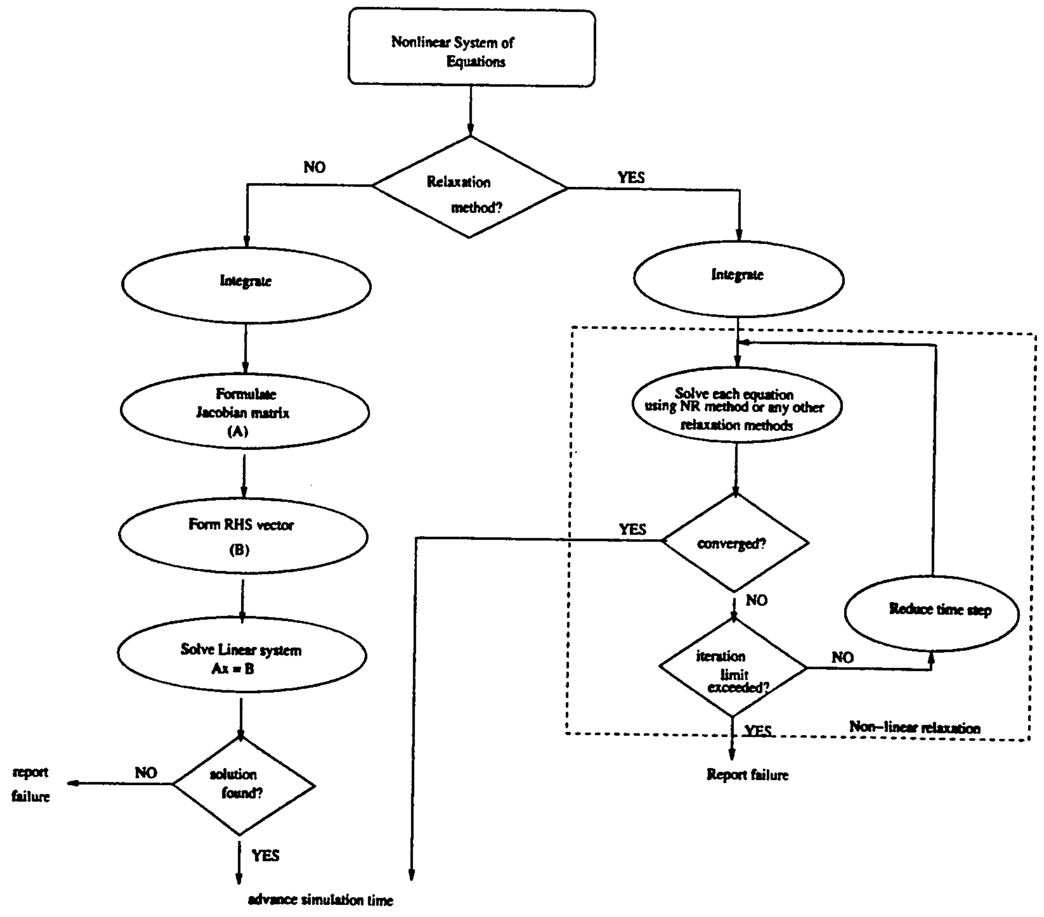


Figure 39: Nonlinear system solution method [34]

Figure 39 shows the two solution steps available in case of a non-linear system. The first method involves integration followed by linearization using any of the well known methods such as Newton-Raphson method or its variants. This reduces the system to a linear algebraic system, which then could be solved using the linear system solution methods discussed in the previous section. The second method for solving the nonlinear system is to use the relaxation methods. Though waveform relaxation (WR) methods have been applied for circuit simulation, they impose strict requirements for convergence [34], and it is not easy to enforce the convergence criteria for general DAEs without the knowledge of the domain they represent. The other form of relaxation is the nonlinear relaxation. It is also called iterated timing analysis (ITA). The method involves integration to convert the nonlinear differential equations to nonlinear algebraic equations and then uses nonlinear solution methods such as Newton method to solve each equation separately, and a relaxation method is used to guarantee that the solutions are mutually consistent. These and the other nonlinear relaxation algorithms can be included in the simulation environment, with very little changes in the kernel.
The approach described above allows multiple solution algorithms to be included in the kernel. A solution method is chosen based on the type of system. The user can also specify the appropriate solution method for the system. This architecture also provides the flexibility and capability to analyze and test the performance of several known algorithms. This approach is easily extensible to accomodate new solution methods and suits a mixed-signal, multi-domain simulation language like VHDL-AMS very well.
In this research work, the above approach is used to chose a linear or a nonlinear solution process. For the linear system solution process, we avoid calculating the Jacobian matrix and thus achieve speedup. Addition of other methods like waveform or nonlinear relaxation have been suggested for future work. As mentioned above, this approach was originally proposed by Vasu [34]. His approach was modified and extended for our investigation. Here are some of the differences:

- Based on the classification of equations into linear and nonlinear, we perform an *incremental matrix build*. The approach by Vasu starts from an empty matrix at every iteration.
- His approach required modeler to specify conservative equations, here we implicitly generate the conservative equations. This is more consistent with the VHDL-AMS language reference manual [18].
- Modified Nodal Analysis (MNA) approach is used to build our matrix whereas he used sparse tableau approach (STA). The matrices generated by MNA are small compared to STA and thus our approach should be more efficient.

6 Matrix Build Optimizations (MBO)

This approach essentially combines all the above approaches into a single approach. The gain obtained from combining the approaches would vary from model to model as all the three approaches might not apply at the same time. However, we will still achieve some improvement as on the minimum, ESO and CEO approach should be applicable to most of the circuits and will provide some speedup.
All the approaches presented above have been implemented in Sierra mixed-signal simulator. Next chapter reports the results of investigating these approaches against a set of models.

6.0.2 Experimental Results of Build Phase Improvements

Evaluation Approach For the purpose of evaluating the various optimization approaches, the results obtained were compared with Sierra Version 1.0. This version of Sierra was modified to incorporate the approaches studied in this thesis. The VHDL-AMS models were then again run with the modified code to do the performance analysis. All the models were executed on a sparc machine running SunOS 5.8 and g++ version 2.95.3 compiler.
The models have been divided into two set of benchmarks: 1) Scalable RLC circuits and 2) Practical circuits. The RLC circuits were used specifically to evaluate scaling and performance while the practical circuits where used to evaluate functionality. Here we give the description of models considered in each set along with its circuit schematic and simulation functional results.

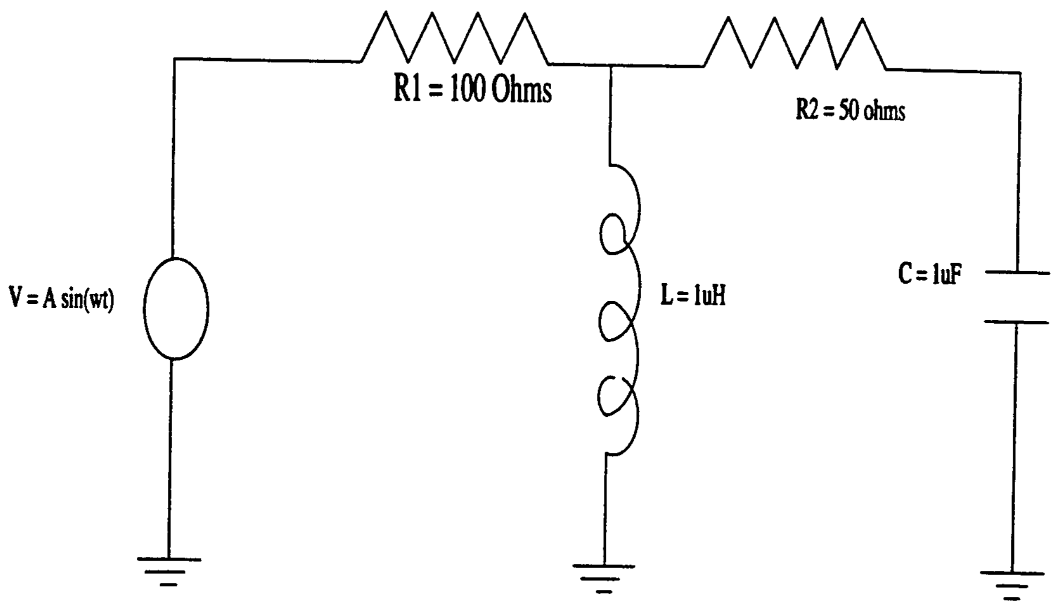


Figure 40: Example scalable circuit for Scaling Study

Scaling Study The models considered in this set are all from a set of similar circuits randomly generated from a general RLC circuit type. These circuits consist of linear and differential-algebraic equations describing resistors, inductors, capacitors and a sinusoidal voltage source. These random circuits were varied in size to generate large models with increasing number of equations. Size was varied between simulation runs. These circuits are static in that the equations do not change during simulation. An example of a small scalable RLC circuit and its VHDL-AMS model is shown in Figure 40 and Figure 41 respectively. Larger models include additional components randomly added in series and parallel combinations. The following equation describes the capacitor :

$$ic = C * vc'dot \tag{41}$$

Here ic and vc refer to the current through and voltage across the capacitor respectively.
The following equation describes an inductor:

$$vl = L * il'dot \tag{42}$$

Here vl and il refer to the voltage across and current through the inductor respectively.
The above equations are broken down into two equations inside the simulator. Thus Equation 41 will be modelled as:

$$ic = C * vc\_dot \tag{43}$$

$$vc\_dot = \frac{dvc}{dt} \tag{44}$$

Six such models were generated for doing the scaling study. Table 5 gives the number of equations in each model and the resulting matrix size for each model. As can be seen from the table the size of matrix for the largest model was 2552 X 2552, which is a considerable large matrix.
We now present the results of the experiments. The approaches referred to by their acronyms:

- Equation Set Optimization (ESO).
- Conservative Equation optimization (CEO).
- Multiple Solution Methods (MSM).
- Matrix Build Optimizations (MBO).

No. of Branch Equations	No. of Terminals	Matrix Size (n x n)
152	50	252
352	100	552
652	200	1052
802	250	1302
1052	350	1752
1552	500	2552

Table 5: Number of Branch equations and the matrix size for the generated RLC models.

```
PACKAGE electricalSystem IS
        NATURE electrical IS real ACROSS real THROUGH
        Ground reference;
        FUNCTION SIN(X : real) RETURN real;
END PACKAGE electricalSystem;
use work.electricalsystem.all;
–entity declaration
ENTITY RLC IS
END RLC;
–architecture declaration
ARCHITECTURE behavior OF RLC IS
        terminal n1, n2, n3 : electrical;
        quantity vs across is through n1;
        quantity vR1 across iR1 through n1 to n2;
        quantity vR2 across iR2 through n2 to n3;
        quantity vL across iL through n2;
        quantity vC across iC through n3;
        constant R1 : REAL := 100.0;
        constant R2 : REAL := 50.0;
        constant C : REAL := 1.0e-6;
        constant L : REAL := 1.0e-6;

        BEGIN
           eqnR1: vR1 == iR1 * R1;
           eqnR2: vR2 == iR2 * R2;
           eqnC : iC == C * vC'dot;
           eqnL : vL == L * iL'dot;
           vsrc : vs == 5.0 * sin ( 2.0 * 3.14 * 1000000.0 *
                              real(time'pos(now))* 1.0e-15);
        END ARCHITECTURE behavior;
```

Figure 41: Instance of a Scalable VHDL-AMS model of the RLC circuit shown in Figure 6.1.

Model Description	Matrix Build Time			Execution Time		
	Sierra v1.0	ESO	Speedup	Sierra v1.0	ESO	Speedup
152	15.51	12.74	17.85	21.54	18.34	14.85
352	50.61	39.75	21.45	75.03	62.66	16.48
652	158.05	114.23	27.72	250.76	198.13	20.98
802	243.57	169.16	30.54	387.53	301.02	22.32
1052	460.93	295.80	35.82	746.37	552.57	25.96
1552	935.01	588.40	37.07	1576.27	1124.01	28.69

Table 6: Matrix Build time, Execution time (seconds) and percentage speedup for Scalable RLC circuits with the ESO approach

Model Description	Matrix Build Time			Execution Time		
	Sierra v1.0	CEO	Speedup	Sierra v1.0	CEO	Speedup
152	15.51	12.98	16.31	21.54	18.97	11.93
352	50.61	37.97	24.97	75.03	61.41	18.15
652	158.05	113.53	28.16	250.76	210.42	16.08
802	243.57	178.90	26.55	387.53	323.97	16.40
1052	460.93	324.38	29.62	746.37	608.20	18.51
1552	935.01	644.24	31.10	1576.27	1282.53	18.63

Table 7: Matrix Build time, Execution time (seconds) and percentage speedup for Scalable RLC circuits with the CEO approach

Model Description	Matrix Build Time			Execution Time		
	Sierra v1.0	MSM	Speedup	Sierra v1.0	MSM	Speedup
152	15.51	12.41	19.98	21.54	18.36	14.76
352	50.61	42.02	16.97	75.03	66.12	11.87
652	158.05	139.13	11.97	250.76	230.45	8.09
802	243.57	219.31	9.96	387.53	360.23	7.04
1052	460.93	424.38	7.92	746.37	709.14	4.98
1552	935.01	888.15	5.01	1576.27	1513.01	4.01

Table 8: Matrix Build time, Execution time (seconds) and percentage speedup for Scalable RLC circuits with the MSM approach

Model Description	Matrix Build Time			Execution Time		
	Sierra v1.0	MBO	Speedup	Sierra v1.0	MBO	Speedup
152	15.51	7.09	54.28	21.54	12.52	41.87
352	50.61	19.54	61.39	75.03	41.60	44.55
652	158.05	58.34	63.08	250.76	144.40	42.41
802	243.57	80.29	67.03	387.53	208.97	46.07
1052	460.93	137.62	70.14	746.37	398.75	46.57
1552	935.01	261.01	72.08	1576.27	792.09	49.75

Table 9: Matrix Build time, Execution time (seconds) and percentage speedup for Scalable RLC circuits with the MBO approach

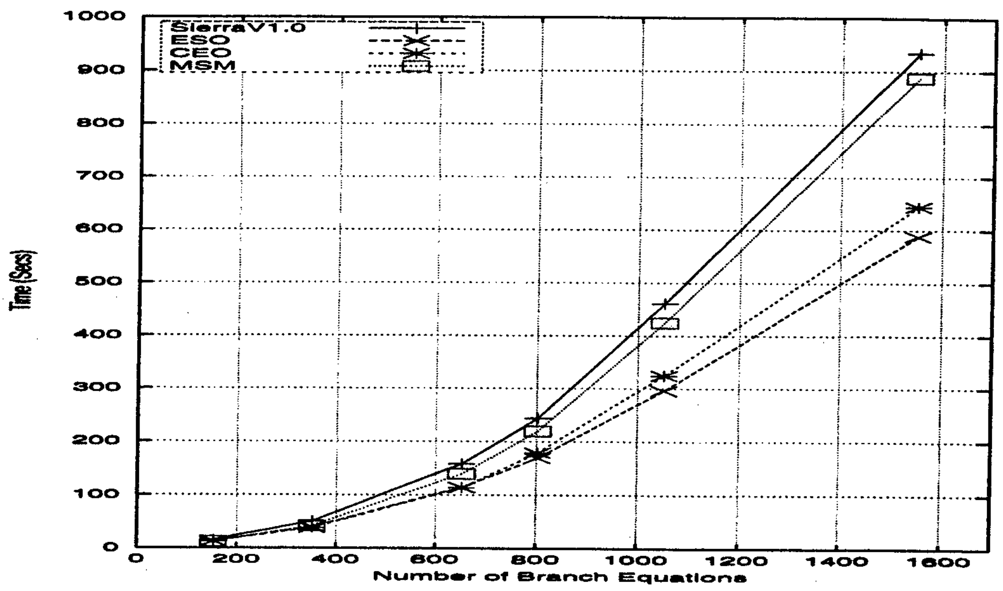


Figure 42: Comparison of Matrix Build Time for approaches ESO, CEO and MSM for the scalable RLC model

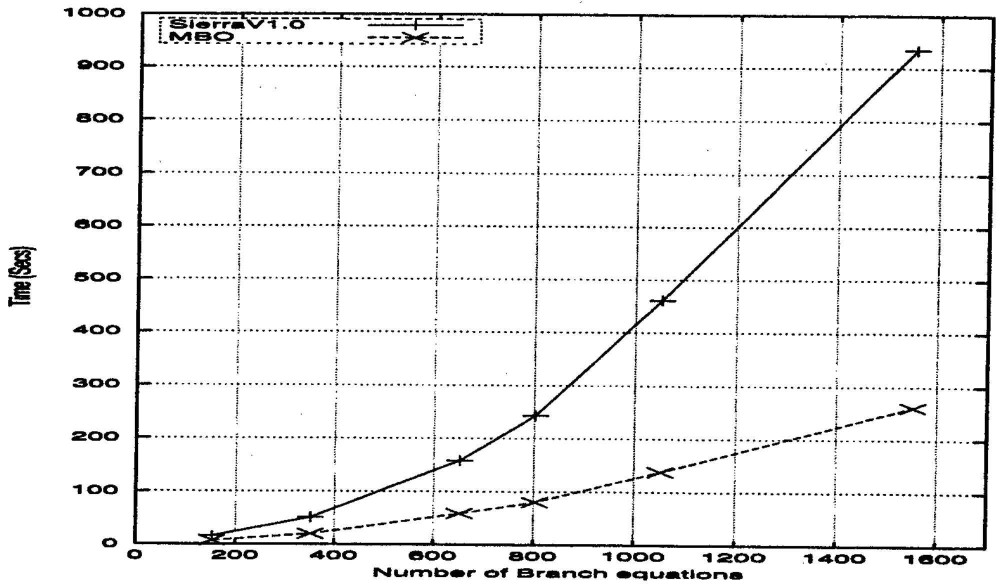


Figure 43: Matrix Build Time for approach MBO for the scalable RLC model

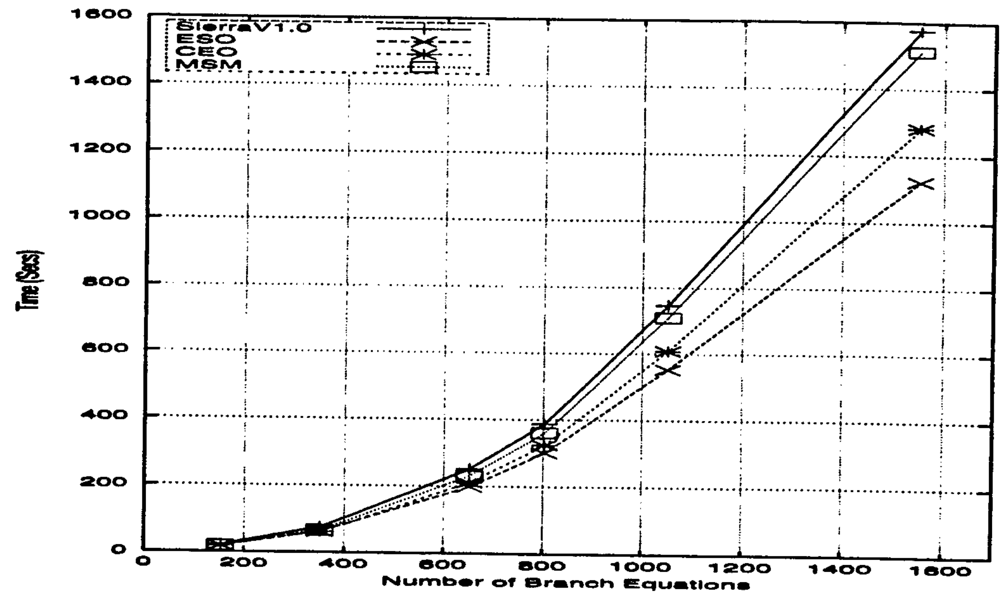


Figure 44: Comparing the total execution times for approaches ESO, CEO and MSM for the scalable RLC model.

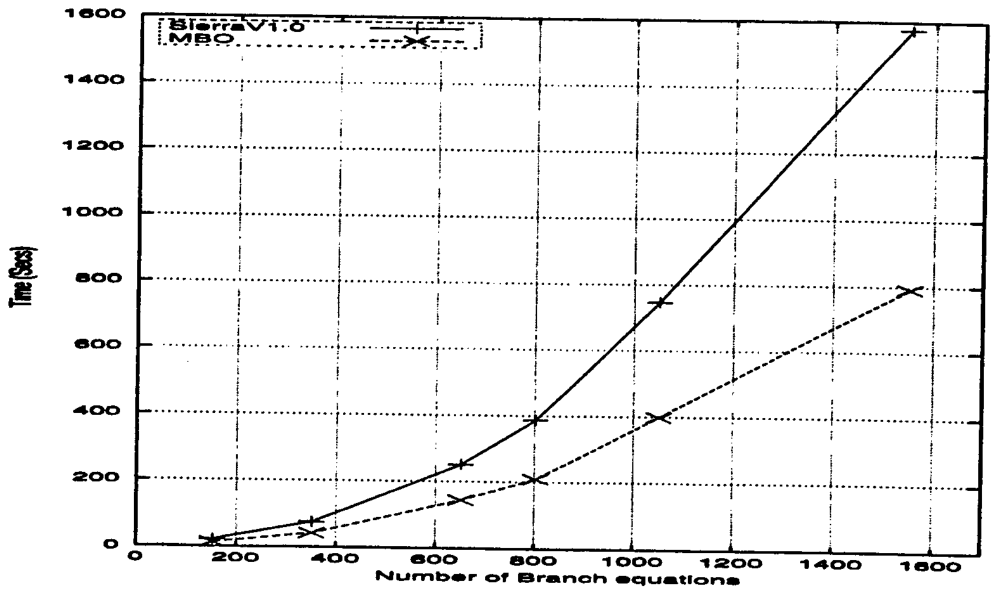


Figure 45: Total Execution Time with approach MBO for the scalable RLC model.

Speedup in matrix build time and total execution time

Tables 6, 7, 8 and 9 show the results obtained. Let us analyze the speedup obtained by each of the approaches.

The following formula is used for calculating the speedup in matrix build phase:

$$\frac{MBT\ in\ Sierra\ V1.0 - MBT\ with\ the\ Current\ approach}{MBT\ in\ Sierra\ V1.0} * 100 \qquad (45)$$

where MBT refers to matrix build time.

Let us first look at the results obtained for the matrix build phase with different approaches. It can be seen that we obtain speedups for all the approaches presented in this research. The speedup obtained with approach ESO falls between 17-38% for the test cases we have considered. The speedup obtained for CEO approach falls between 16-32%, while for the MSM approach it falls between 5-20% for the scalable RLC models. Finally the MBO approach which combines all the above three approaches gives a speedup that falls in the range of 54-73%.

Let us now look at the results obtained for the total execution time with different approaches.

The following formula is used to calculate the speed up :

$$\frac{TET\ in\ Sierra\ V1.0 - TET\ with\ the\ Current\ approach}{TET\ in\ Sierra\ V1.0} * 100 \qquad (46)$$

where TET refers to Total Execution Time.

Again, as can be seen from the tables, we obtain speedup in total execution time for all the approaches. The ESO approach gives a speedup between 14-29%, CEO between 11-19%, MSM between 4-15% and finally the speedup for MBO approach falls between 40-50%. The speedups obtained are really significant. This research is concentrated on improving the simulation speed through optimizing matrix build phase. Since matrix build typically occupies 50-60% of the total execution time, a reduction in the time taken in this phase would definitely lead to reduction in the execution time. However, the speedup obtained in total execution time would be less as compared to that obtained in matrix build time. This is evident as the matrix build time is only some portion of the total execution time. The rest of the time goes into solving the matrix and doing other operations for simulation. So, the overall speedup would be less than the speedup obtained from improving a single portion of execution time. This can clearly be seen from the results obtained also as the speedup obtained in matrix build time is more than the total execution time.

Clearly the ESO and CEO approach produce better results when compared with MSM approach. ESO approach performs slightly better than CEO approach. It clearly outlines the fact that ESO approach, that divides the set into a base and a auxiliary set saves a considerable time during the setup phase. Also CEO approach optimizes loading of conservative equations and the speeup obtained shows that this approach also gives significant speedup. It can thus be infered that loading KCL equations at every time point is very time consuming and preserving the enteries in the matrix corresponding to KCL equations between succesive timepoints is definitely a good approach. MSM approach has a lot of overhead as it first classifies the equations into linear and nonlinear and then determines the equation set type at every timepoint and then switches between different solvers. This explains why the speedup seen with the MSM approach is less, but still 5-20% is also very significant when it comes to mixed-signal simulation. Also when all the three approaches are combined in approach MBO, we really observe a significant speedup of **54-73%** in the matrix build time and **40-50%** in the total execution time. Figures 43 and 45 show the improvements obtained by incorporating the individual with the MBO approach.

References

[1] Acuna, E. L., Dervenis, J. P., Pagones, A. J., YANG, F. L., and Saleh, R. A. Simulation techniques for mixed analog/digital circiots. *IEEE Journal of Solid-State Circuits 25*, 2 (April, 1990), 353-363.

[2] Aho, A. V., Sethi, R., and Ullman, J. D. *Compilers, principles, techniques and tools.* Addison-Wesley, Reading, MA, 1986.

[3] Antrim Design Systems, Inc. Advanced Techniques for the Simulation of Mixed-Signal Integrated Circuits, 1999.

[4] Ashenden, P. J. *The designer's guide to VHDL.* Morgan kaufmann, 1995

[5] Bakalar, K., and Christen, E. VHDL 1076.1: The Design of a language extension for VHDL. In *VHDL: The next 10 years* (April, 1997), pp. 119-127.

[6] Bapat, S. Large Scale Evaluation of VHDL-AMS simulator using scalable generic VHDL-AMS models, Master's thesis (to be submitted), University of Cincinnati, 2002.

[7] Buchanan, J. L., and Turner, P. R. *Numerical methods and analysis.* McGraw Hill, 1992.

[8] Cellier, F. E. *Continuous system modeling.* Springer-Verlag, 1991.

[9] Chandy, K. M., and Sherman, R. Space-time and simulation. In *Distributed Simulation* (1989), Society for Computer Simulation, pp. 53-57.

[10] Chetput, C.L. An anlog kernel using the direct method for solving ordinary differential algebraic equations in a mixed-mode simulator. Master's thesis, University of Cincinnati, 1997.

[11] Fahrland, D.A. Combined discrete-event continuous systems simulation. *Simulation*, 1970.

[12] Frey, P. Parallel synchronization of continuous time discrete event simulators. In *Proceedings of the International Conference on Parallel Processing* (August 1997), pp. 227-231.

[13] Frey, P., Nellayappan, K., Mayiladuthurai, R. S., Chandrashekar, C. L., Carter, H. W. SEAMS: Simulation Environment for VHDL-AMS. In *Proceedings of the 1998 Winter Simulation Conference.* (1998), pp. 539-546.

[14] Frey, P., and Carter, H. W. Formal specification and verification of optimistic simulation protocols. Tech. Rep. TR218/09/ECECS, University of Cincinnati, 1998.

[15] Gielen, G., Wambacq, P., and Sansen, W. M. Symbolic analysis methods and applications for analog circuits: A tutorial overview. In *Proceedings of the IEEE* (February 1994), vol. 82, pp. 286-303.

[16] Griewank, A., Juedes, D. and Utke, J., *ADOL-C: A Package for the Automatic Differentiation of Algorithms written in C/C++*, Germany, 1996.

[17] Ho, C.W., Ruehli, A.I., and Brennan, P.A., *The modified nodal approach to network analysis*, IEEE Trans. on Circuits and Systems, 1975, Vol. CAS-22, No. 6, pp. 504-509.

[18] IEEE*Standard VHDL Language Reference Manual (Integrated with VHDL-AMS changes*, 1998.

[19] Jennings, A., McKeown, J. J. *Matrix computation.* John Wiley & Sons, New York, NY, 1992.

[20] Kundert, K. S. Sparse matrix techniques, In *Circuit Analysis, Simulation and Design* , A. Ruehli, Ed. North-Holland, 1986.

[21] Kundert, K. S. *The Designer's Guide to SPICE and SPECTRE.* Kluwer Academic Publishers, 1995.

[22] Kundert, K. S. *Sparse User's Guide*, University of California, Berkeley, 1998.

[23] Kielkowski, R. *Inside SPICE.* McGraw-Hill, 1995.

[24] Lightner, M. *Computer aided circuit simulation.* CRC Press, 1993.

[25] Litovski, V. and Zwolinski, M. *Circuit Simulation and Optimization.* Chapman and Hall, 1997.

[26] MANAVALAN J. K. A. Performance evaluation and speed improvement of the SEAMS VHDL-AMS simulator using dynamic adjustment of the analog simulation interval. Master's thesis, University of Cincinnati, 1998.

[27] MARON, M.J. *Numerical analysis.* Macmillan Publishing Co., Inc., 1982.

[28] MAYILADUTHURAI, R. S. Processing discontinuities and SPICE modeling in VHDL-AMS. Master's thesis, University of Cincinnati, 1998.

[29] NAVABI, Z. *Analysis and modeling of digital systems.* McGraw-Hill, 1993.

[30] NELLAYAPPAN, K. Seams: A mixed-signal simulation environment for vhdl-ams with emphasis on run-time elaboration and analog design partitioning. Master's Thesis, University of Cincinnati, 1998.

[31] PANDEY, S. Improving performance of mixed-signal simulation by reducing equation-set. Master's thesis, University of Cinncinati, 2002.

[32] PARR, T. J.*Language translation using PCCTS and C++: a reference guide.* Automata Publishing Company, San Jose, CA, USA, Jan. 1997.

[33] SALEH, R. A., ANTAO, B. A. A., AND SINGH, J. Multilevel and mixed-domain sumulation of analog circuits and systems. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems 15* (January 1996 , pp. 68-81. Feature Article.

[34] SHANMUGASUNDARAM, V. *A dynamic multiple-solution approach to improve the efficiency of VHDL-AMS simulation.* Master's thesis, University of Cincinnati, 1998.

[35] VLACH, J., AND SINGHAL, K. *Computer methods for circuit analysis and design.* Van Nostrand Reinhild, New York, NY, 1994.

[36] WILLIS, J., WILSEY, P.A., MARTIN, D. A., CHETLUR, M., NEWSHUTZ, R., CARTER, H. W., SHANMUGASUNDARAM, V., AND ROGERMAN, S. *Advanced Intermediate Representation with Extensibility (AIRE).* Available on the world wide web at `http://www.vhdl.org/aire`

[37] WILSEY, P.A., MARTIN, D.E., AND CHAWLA, P. SAVANT: An extensible object-oriented intermediate for VHDL. In *VHDL User's Group Spring 1996 Conference* (Mar. 1996), pp. 275-281.

[38] XIA, E. Z., AND SALEH, R. A. Parallel waveform-newton algorithms for circuit simulation. *IEEE Transactions on Computer Aided Design 11* (April 1992), 432-442. Feature Article.

[39] YE, L. *A language and Elaboration Method to Support Mixed-signal Modeling and Simulation.* PhD thesis, ECECS Department, University of Cincinnati, 1996.

[40] ZEIGLER, B. P. *theory of modeling and simulation.* John Wiley & Sons, Inc., New York, NY, 1976.

[41] A guide to mixed-signal simulation. Technical report, Analogy Corporation, 1995.

[42] H.M. Dietel and P.J. Dietel. *C++ How to program.* Prentice Hall, New Jersey, 2001.

[43] Juedes D. Griewank, A. and J Utke. Adol-c: A package for the automatic differentiation of algorithms written in c/c++. *ACM TOMS*, 1996.

[44] R.M. Kielkowski. *Inside Spice.* McGraw-Hill, 1998.

[45] H. Kopka and P.W. Daly. *A Guide to LATEX.* Addison-Wesley, 1999.

[46] V. Litovski and M. Zwolinski. *VLSI Circuit Simulation and Optimization.* Chapman and Hall, New York-10003, 1997.

[47] J.K.A. Manavalan. Performance evaluation and speed improvement of the seams vhdl-ams simulator using dynamic adjustment of the analog simulation interval. Master's thesis, University Of Cincinnati, 1998.

[48] K. Nellayappan. A mixed-signal simulation environment for vhdl-ams with emphasis on run-time elaboration and analog design partitioning. Master's thesis, University Of Cincinnati, 1998.

[49] D. O. Pederson. A historical review of circuit simulation. *IEEE Transactions on Circuits and Systems CAS-31*, January 1984.

[50] Thomas Linwood Quarles. *Analysis of Performance and Convergence Issues for Circuit Simulation.* PhD thesis, University of California, Berkeley, April 1989.

[51] Klimeck G. Salazar-Lazaro C. Keymeulen D. Stoica, A. and A. Thakoor. Evolutionary design of electronic devices and circuits. IEEE Press., December 1999.

[52] Subramani, K. The Design of Parallel VHDL Simulation Kernel Based on Time Warp. Master's Thesis, University of Cincinnati, February 1998.

AFRL-IF-WP-TM-2004-1535